PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06T 11/00	A1	(11) International Publication Number: WO 95/31794
		(43) International Publication Date: 23 November 1995 (23.11.95)

(21) International Application Number: PCT/US95/06237

(22) International Filing Date: 17 May 1995 (17.05.95)

(30) Priority Data:
08/245,049 17 May 1994 (17.05.94) US

(71) Applicant *(for all designated States except US)*: APPLE COMPUTERS, INC. [US/US]; One Infinite Loop, Cupertino, CA 95014 (US).

(72) Inventors; and
(75) Inventors/Applicants *(for US only)*: SWEN, Iue-Na (Steve) [CN/US]; 22395 St. Andrews Avenue, Cupertino, CA 95014 (US). STOKES, Michael, D. [US/US]; 22462 Salem Avenue #3, Cupertino, CA 95014 (US). MOHR, Thomas, E. [US/US]; 555 NW Park #407, Portland, OR 97209 (US).

(74) Agent: PETERSON, James, W.; Burns, Doane, Swecker & Mathis, P.O. Box 1404, Alexandria, VA 22313-1404 (US).

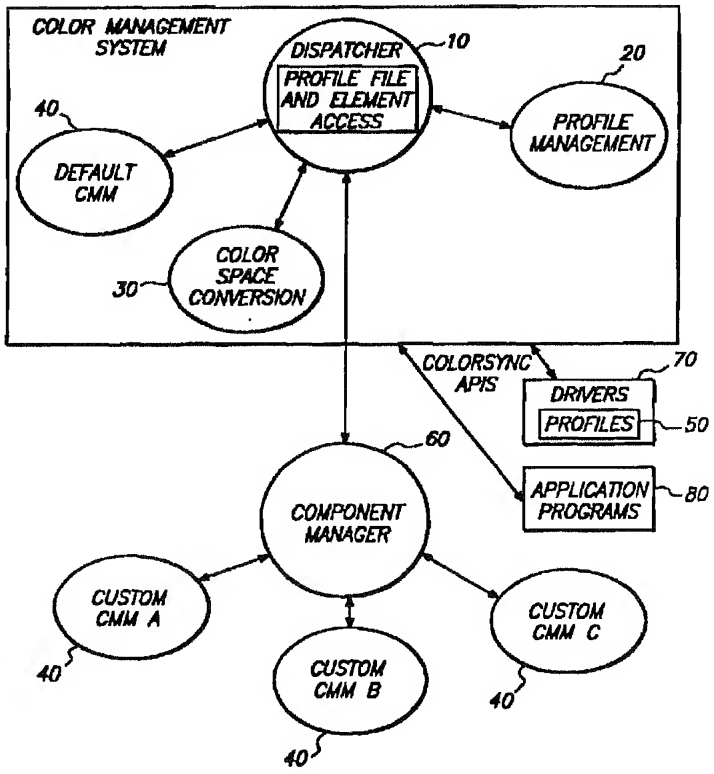
(81) Designated States: AM, AT, AU, BB, BG, BR, BY, CA, CH, CN, CZ, DE, DK, EE, ES, FI, GB, GE, HU, IS, JP, KE, KG, KP, KR, KZ, LK, LR, LT, LU, LV, MD, MG, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, TJ, TM, TT, UA, UG, US, UZ, VN, European patent (AT, BE, CH, DE, DK, ES, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG), ARIPO patent (KE, MW, SD, SZ, UG).

Published
With international search report.
Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: COMPUTER GRAPHICS COLOR MANAGEMENT SYSTEM

(57) Abstract

An operating-system-level color management system provides a scalable, flexible and extensible solution to managing color in color computer graphics systems. A disk-based tagged-element structure allows selective access to profile data. The color management system dynamically arbitrates and dispatches to code modules to perform color management functions such as color matching, color space conversion, profile management, profile file and element access, profile validation, and converting profiles to PostScript. A CMM arbitration method ensures fairness for both source and destination profiles.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AT	Austria	**GB**	United Kingdom	**MR**	Mauritania
AU	Australia	**GE**	Georgia	**MW**	Malawi
BB	Barbados	**GN**	Guinea	**NE**	Niger
BE	Belgium	**GR**	Greece	**NL**	Netherlands
BF	Burkina Faso	**HU**	Hungary	**NO**	Norway
BG	Bulgaria	**IE**	Ireland	**NZ**	New Zealand
BJ	Benin	**IT**	Italy	**PL**	Poland
BR	Brazil	**JP**	Japan	**PT**	Portugal
BY	Belarus	**KE**	Kenya	**RO**	Romania
CA	Canada	**KG**	Kyrgystan	**RU**	Russian Federation
CF	Central African Republic	**KP**	Democratic People's Republic of Korea	**SD**	Sudan
CG	Congo			**SE**	Sweden
CH	Switzerland	**KR**	Republic of Korea	**SI**	Slovenia
CI	Côte d'Ivoire	**KZ**	Kazakhstan	**SK**	Slovakia
CM	Cameroon	**LI**	Liechtenstein	**SN**	Senegal
CN	China	**LK**	Sri Lanka	**TD**	Chad
CS	Czechoslovakia	**LU**	Luxembourg	**TG**	Togo
CZ	Czech Republic	**LV**	Latvia	**TJ**	Tajikistan
DE	Germany	**MC**	Monaco	**TT**	Trinidad and Tobago
DK	Denmark	**MD**	Republic of Moldova	**UA**	Ukraine
ES	Spain	**MG**	Madagascar	**US**	United States of America
FI	Finland	**ML**	Mali	**UZ**	Uzbekistan
FR	France	**MN**	Mongolia	**VN**	Viet Nam
GA	Gabon				

COMPUTER GRAPHICS COLOR MANAGEMENT SYSTEM

BACKGROUND OF THE INVENTION

Field of the Invention

The present invention relates to color computer graphics, more particularly to color management systems, i.e., systems (typically software) that provide for the exchange and use of color information in a computer environment in which different devices have different color characteristics.

State of the Art

As color devices for input and output proliferate, so do the problems of moving images between them with good results. Differing device types operate in different color spaces. Within those spaces, the colors the devices can actually produce constitute differing gamuts, or ranges of colors. Monitors, for example, typical display colors as combinations of red, green, and blue, and are said to work in the RGB color space. Monitors by different manufacturers may be capable of displaying different intensities of those three colors, so that even though they work in RGB space, their gamuts may be different. Printers typically work with varying intensities of cyan, magenta, yellow, and black, and are said to work in the CMYK color space. Print technologies vary drastically, and the gamut that an ink jet color printer can display may be quite different from one based on another technology. A single printer may be able to produce different gamuts depending on the paper or ink being used at the time of printing. Devices with differing color gamuts cannot reproduce each other's colors exactly but shifting the colors of one by a carefully chosen profile may improve the visual match between them.

The most obvious difference for many users is between an image as it appears on the monitor and as printed by a color printer. Monitor colors are additive--adding color moves the image toward white--and are generally more vivid. Printed colors are subtractive--adding colors moves the image toward black or dark gray. Under most viewing conditions, the white of a monitor will

appear bluish when compared to a sheet of paper, even though the eye perceives both as white when viewed separately.

A color management system is a system that supports color information exchange, color matching, color profile management, and device calibration. Color management offers the means of transmitting color images and documents containing color across local and wide area networks while maintaining the fidelity of the colors of the original image or document.

In January 1993, Apple Computer introduced a color management system, ColorSync™ 1.0, in the form of a suite of utilities. The ColorSync Utilities define a process (the color matching method, or CMM) for matching colors between gamuts, and a data structure (the color profile) for holding information about the characteristics of a particular gamut. The ColorSync Utilities provide a default CMM and a default system color profile for Apple monitors. Because the ColorSync Utilities are designed to provide an "open system" for color management, developers can use an Apple-supplied default CMM and color profile, create their own, or obtain them from vendors, and use them with the ColorSync Utilities.

The ColorSync Utilities provide applications with several tools for matching colors between devices. These include: a default system color profile that describes the gamut of the Apple RGB 13 inch monitor, a control panel (cdev) interface by which users can set their system profile, a means of specifying and obtaining color profiles for other devices, a means of tagging pictures with color profiles, a default color matching method, a folder for storing color profiles, an open architecture that allows developers to create or obtain a custom color matching method and associate it with a device profile, and the ability to report matching progress to the user.

The ColorSync Utilities contain high level routines that can query and set a color profile, begin and terminate matched drawing, match a picture's colors as it is drawn, tag a picture with one or more color profiles, and test whether colors are in a device's gamut. The ColorSync Utilities also contain low-level routines for matching specific lists of colors, matching pixel map colors,

monitoring the progress of color matching, and creating a custom color matching method.

Color matching means converting colors between differing color gamuts. A color matching method (CMM) implements an algorithm that determines how to convert the colors. In general, whereas device drivers supply CMMs appropriate to their device (since the driver knows the color characteristics of the device), applications typically do not need to create CMMs. With ColorSync™, device drivers can create their own CMMs, obtain them from other vendors, or use the CMM provided as the default by Apple.

Because enumerating all the colors in a gamut is generally impractical, the ColorSync Utilities describe a color gamut by a set of characteristics (the color profile) that define key parameters, such as the points of purest white and black, tonal response curves for primary colors, and the type of CMM for which the profile was intended.

Each CMM may offer matching options, so that the user can tailor the match depending on the kind of image being processed. Typically, color matching alters the colors of an image being transferred from one device to another such that the resulting image looks as close to the original as possible. A number of methods may be used to alter the colors.

Given two overlapping but non-coincident color gamuts A and B, for example, in some circumstances it may be best to only alter the colors of space A that do not exist in space B. The advantage is that many colors are exactly the same in both images, which may be required when using Pantone™ colors to prepare a book for printing, for example. The disadvantage is that the best mapping of the colors that must change may be the same as existing colors, reducing the actual number of colors in the copy. Maintaining the same colors wherever possible is called a colorimetric match.

With realistic, photographic images it may be better to alter all the colors of the first space, to move them so that their relative positions are the same when relocated to the second space. The advantage is that a greater number of colors may be preserved, and the relationship between colors is maintained. The

disadvantage is that none of the original colors is the same in the copy. Since the colors in this case will appear the same to the eye, even though they may vary drastically if measured by instrument, this is called a perceptual match.

A third possibility is to maintain saturation: computer-generated bar charts, for example, do not require exact matching of the particular hues used; it may be more important that the image is vivid so it will project well.

Applications can set one of three corresponding matching options in a source profile header to choose variations on the techniques used for color matching. A perceptual match uses the destination white point, maintains lightness, and is good for real-world images. A colorimetric match uses the source white point and is good for spot-color matches. A saturation match uses the destination white point, maintains saturation for out-of-gamut colors, and is good for charts.

ColorSync allows multiple data types that have different matching to exist on the same page. For example, a page might contain a photograph that was scanned, some bar charts that were generated with a statistical package, and some line art that was generated using spot colors, as shown in Figure 1.

When the page is captured as a picture, the data stream will represent different matching operations required for proper rendering, as follows:

...

// For the photograph //

PicComment(CMBeginProfile)

{Contains pr2, a profile which has a device type of 'scnr,' and options that indicate perceptual match}

Here matching occurs using the perceptual match option.

PicComment (CMEndProfile)

Here any matching occurs using the system and device profiles.

...

//For spot color line art//

PicComment (CMBeginProfile)

{Contains pr1, a profile which has device type 'mntr' and options that

indicate colorimetric match}

...

Since no CMEndProfile picture comment was encountered, matching continues using this profile.

...

PicComment (CMDisableMatching)

Here there is no matching and data is passed straight through.

PicComment (CMEnableMatching)

Now things are matched using the 'mntr' profile with colorimetric matching again.

...

//For bar chart//

PicComment (CMBeginProfile)

{Contains a profile which has device type 'mntr' and options that indicates saturation}

Here matching occurs using the saturation match option.

PicComment (CMEndProfile)

...

Here any matching occurs using the system and device profiles again.

An application or driver sets the options to the values it needs before calling ColorSync Utilities' matching functions, or before tagging a picture with a color profile (i.e., embedding the color profile in the picture).

All CMMs are components that a Component Manager of the Macintosh system software calls in response to calls of applications or drivers through a defined application interface (API). A component is a piece of code that provides a defined set of services to one or more clients. Applications, system extensions, as well as other components can use the services of a component. A component typically provides a specific type of service to its clients. For example, a component might provide image compression or image decompression capabilities; an application could call such a component, providing the image to compress, and the component could perform the desired

operation and return the compressed image to the application. In the case of ColorSync, the type of service performed is color matching. An application or device driver can call ColorSync, providing an image, a source color profile and a destination color profile, and ColorSync will call the appropriate CMM component to perform color matching or other color management functions and return the result (e.g., a color-matched image) to the application or device driver.

Multiple components can provide the same type of service. For example, separate components might exist that can compress an image by 20 percent, 40 percent, or 50 percent, with varying degrees of fidelity. Similarly, different CMMs may achieve varying degrees of color fidelity. All components of the same type must support the same basic interface. This requirement allows an application or driver to use the same interface for any given type of component and get the same type of service, yet allows applications and drivers to obtain different levels of service.

The Component Manager provides access to components and manages them by, for example, keeping track of the currently available components and routing requests to the appropriate component. The Component Manager classifies components by three main criteria: the type of service provided, the level of service provided, and the component manufacturer. The Component Manager uses a component type, consisting of a sequence of four characters, to identify the type of service provided by a component. CMMs have a component type of `CMM ' `. In addition, CMMs have a signature. The signature of the default CMM is `appl' `. Third party CMMs are registered with Apple and assigned a CMM signature.

The Component Manager provides services that allow applications to obtain run-time location of and access to functional objects by creating an interface between components and clients. Instead of implementing support for a particular data format, protocol, or model of a device, a standard interface is used through which an application can communicate with all components of a given type, such as CMMs. The Component Manager may be used to locate

and communicate with components of that type. Those components, in turn, provide the appropriate services to the client application.

Given a particular component type, the Component Manager can locate and query all components of that type. An application can find out how many components of a specific type are available and can get further detail about a component's capabilities without having to open it first. For each component, the Component Manager keeps track of many characteristics, including its name, icon, and information string, which may contain any arbitrary information about the component.

Components of the type 'CMM' provide color matching services. All components of type 'CMM' share a common application interface, but each component may support a unique color matching technique or take advantage of a special hardware implementation. Individual components may support additions to the defined application interface, as long as they support the common routines any algorithm-dependent or implementation-dependent variations of the general interface can be implemented by each 'CMM' component as extensions to the basic interface.

The Component Manager allows a single component to service multiple client applications at the same time. Each client application has a unique access path to the component. These access paths are called component connections. A component connection is identified by specifying a component instance. The Component Manager provides this component instance to an application when it opens a connection to a component. This component maintains separate status information for each open connection.

Multiple applications might each open a connection to a color matching component. The Component Manager routes each application request to the component instance for that connection. Because a component can maintain separate storage for each connection, application requests do not interfere with each other and each application has full access to the services provided by the component. Further information concerning component-based design may be found in *Inside Macintosh*, More Macintosh Toolbox.

In accordance with the foregoing component-based design, the ColorSync Utilities provide a default CMM; developers of device drivers can create or buy other CMMs, tailored to the requirements of their devices.

Referring more particularly to Figure 2, the shaded blocks represent the color management system, which includes a component based color management framework interface, a default CMM, and Apple supplied profiles including a default system profile and additional profiles of Apple peripherals. The color management system is interoperable with third party CMMs and with third party device profiles, which the color management system may use to perform color matching. The color management system may be called upon by applications directly (through a predefined application program interface, or API) to perform color matching, or the application may call upon a graphics library or an imaging library to perform color-matched services, in which case the graphics library or the imaging library will then call upon the color management system to perform color matching.

To perform color matching, the ColorSync Utilities use the Component Manager to call a color matching method. The actual matching happens inside the color matching method component. Figure 3 shows the relationship between an application, the ColorSync Utilities, and the color matching component.

Device drivers generally provide their own color profiles, and perform the match by calling the ColorSync Utilities. This ensures that as new devices become available, existing applications continue to receive matched input and produce matched output. Applications can print with confidence that the device driver will produce the best color translation.

Applications can use color matching to create a print preview dialog, showing which colors in the printed image are out of the image gamut as it appears on the monitor, for example. Applications that generate images can

automatically tag the image with the appropriate profile, or allow the user to tag it.

Color matching always matches the colors of a source profile to the colors of a destination profile using a color matching method. The matching process takes one or more steps, depending on whether the source and destination profiles are the same type, and therefore use the same CMM.

If the source and destination profiles can use the same color matching method, the colors are mapped directly from one color gamut to the other.

If they use different methods, colors are mapped from the source profile to a profile describing XYZ space, a standard intermediate space, using the matching method associated with the source profile. Colors are then mapped from XYZ space to the destination profile, using the matching method associated with the destination profile. If either the source or the destination profile parameter to a matching routine is NIL, the ColorSync Utilities use the default system profile in its place.

Figure 4 diagrams the default condition, in which input devices match to the system color profile, and output devices match from it to the profile of their device.

In the input matching step, colors are mapped from the device's profile to the RGB space of the system color profile. In the output matching step, the colors are mapped from the system RGB space to the color space of the output device.

Thus, in this default matching case, input drivers match to the system profile, and output drivers match from the system profile. Applications work without modification since the drivers do the matching.

Profiles reside in files (usually in a ColorSync™ Profiles folder), with device drivers, or in pictures. All ColorSync™ 1.0 profiles have a header, a copy of an Apple CMProfileChromaticities record (containing colorant values), profile response data for the associated device, and a profile name string for use in dialog boxes. Custom profiles may also have additional, private data. The profile record is diagrammed in Figure 5.

The device type field indicates the type of device for which the profile is intended. Three standard types are:

PrinterProfile 'prtr'

ScannerProfile 'scnr'

MonitorProfile 'mntr'

The response data fields contain nine curve tables defining device gamma values. Seven tables are always filled in; the last two are used only for printers using four colors (CMYK). The first table is for grayscale values. The next three are red, green, and blue values, followed by three for cyan, magenta, and yellow values. Each table includes a count field. The count value specifies the number of entries in the curve table except as follows:

when count is 0, then a linear response (slope equal to 1.0) is assumed,

when count is 1, then the data entry is interpreted as a simple gamma value (ranging from 0 to 8), and

when count is 3, the entries are interpreted as a gamma-offset-gain model.

The eighth and ninth tables are for CMYK printers and specify undercolor removal and black generation data, respectively. Placing a 0 in the counts field specifies maximum undercolor removal and maximum black generation.

The name string of the profile is used by the user interface. Applications can retrieve ColorSync 1.0 profiles using a GetProfile function. Calibration programs can attach a profile to a device using a SetProfile function. Profiles can be obtained from currently attached devices that support ColorSync and from the ColorSync™ Profiles folder, if present. Profiles can be created by hand, or by using a utility such as ProfileMaker.

The ColorSync Utilities provide an open architecture that can work with a variety of color matching methods and color profiles. ColorSync determines which CMM to use, based on the `CMMType` field of the source and destination profiles used in a match. (If either profile is missing or NIL, ColorSync uses the system profile.)

Since any particular configuration may include a variety of input and output devices, each with its own CMM and profiles, the ColorSync Utilities follow a defined sequence to determine which CMM to use in a given match. In the sequence, the default CMM is last in the list, to be used if no other CMM is specified, or a specified CMM is not available. The process of choosing a CMM to perform a match is referred to as CMM arbitration.

ColorSync™ 1.0 chooses which CMM to use for a match according to this sequence:

1. If both the source and destination profiles have the same `CMMType` and the corresponding CMM component is available then matching is performed entirely by that CMM, as shown in Figure 6, where RGB input data is matched to CMYK output data. (If the CMM is not available, ColorSync uses the default CMM.)

If the profiles have different `CMMType` values then ColorSync attempts the following alternatives in order.

2. If the CMM for the destination profile is available (and is not Apple's) then it is called with the two profiles. If the CMM can match using that source profile it returns `noErr` and matching is performed entirely by the CMM corresponding to the destination profile. If a profile error is returned then alternative 3 is tested.

The Apple-supplied default CMM will always respond `noErr` and try to perform a match. All profiles must contain the elements necessary for the default CMM. If either the source or destination profile has a `CMMType` value other than 'appl,' that CMM will get a chance to perform the match. The default CMM is always the last to be chosen, and it always attempts the match if asked.

3. If the CMM for the source profile is available (and it is not the default CMM) then it is called with the two profiles. If it can perform the match then matching is performed entirely by the CMM named by the source profile.

If the CMM returns an error then the ColorSync Utilities try alternative 4.

4. If the CMMs for both the source profile and destination profile are available--but neither can perform the match, as determined by steps 2 and 3--ColorSync converts the source data from RGB space to XYZ space using the source profile and its corresponding CMM. Then ColorSync converts the XYZ color space to the destination RGB space using the destination profile and its CMM, as shown in Figure 7.

If this is not possible (one of the CMMs is missing) then alternative 5 is the fallback.

5. If one or both of the CMM components corresponding to a profile is not available then ColorSync uses the default CMM for that step of the match, using the Apple format data required in all ColorSync profiles.

A CMM must support RGB and CMYK as input color spaces, and RGB, XYZ, and CMYK as output color spaces.

In the foregoing CMM arbitration method, the potential exists for some CMMs to be unfairly shut out. Since the destination CMM is queried first and given first opportunity to perform the color match, the situation may arise in which a destination CMM may aggressively respond affirmatively to all queries, even if it is not the best CMM for the job.

Furthermore, in ColorSync™ 1.0, no API support is provided for PostScript® conversion. PostScript has established itself as the page description LANGUAGE of choice among graphics and pre-press user communities. It is also becoming one of the standard imaging models supported on all major platforms. Integrating color management system functions to support PostScript is therefore highly desirable.

Most high-end printers and image setters have PostScript controllers and do not understand any command other than PostScript. To control the color output from these devices, PostScript Level 2 defined a device-independent color mechanism to specify and render color. To integrate color device profiles used by color management systems with PostScript devices, conversions have to

be done between color profiles and PostScript.

The main function of a PostScript driver is to convert from the platform native graphic format to PostScript for printing. One example of a PostScript driver is the PSWriter 8.1. It converts Quickdraw format on the Macintosh to Postscript. In the ColorSync 1.0 system, when a color profile is embedded in the document, the driver will convert it into Postscript also. If no profile is embedded with the document, the ColorSync System Profile is used.

The ColorSync 1.0 profile format is extendible, as are some other color profile formats. Many profile vendors therefore put custom information in the private fields of such formats. The custom data provides higher performance when used with the preferred CMM. However, the driver does not understand how to interpret the custom data and may not have access to the preferred CMM. As a result, the quality of the output from the Postscript devices is less than optimal.

ColorSync™ 1.0 makes no provision for profile validation to determine whether or not a profile contains all of the required fields. Using profiles that do not contain all of the required fields may have a range of effects from mild (improper matching) to severe (causing a system crash).

ColorSync™ 1.0 provides only limited support for concatenation profiles in which a transformation is calculated from a sequence of profiles (for a print preview operation, for example, from monitor device space to printer device space and back to monitor device space). In ColorSync™ 1.0, concatenation profiles cannot be saved for future use or embedded in images.

Moreover, in ColorSync™ 1.0, the profile format is memory resident. Such a profile format does not readily lend itself to extension and improvements without creating a memory burden on the host system.

SUMMARY OF THE INVENTION

The present invention, generally speaking, provides in an operating-system-level color management system a scalable, flexible and extensible solution to managing color in color computer graphics systems. A

disk-based, tagged-element structure allows selective access to profile data. The color management system dynamically arbitrates and dispatches to color matching modules (CMMs) and other code modules to perform color management functions such as color matching, color space conversion, profile management, profile file and element access, profile validation, and converting profiles to PostScript. A CMM arbitration method ensures fairness for both source and destination profiles.

BRIEF DESCRIPTION OF THE DRAWING

The present invention may be further understood from the following description in conjunction with the appended drawing. In the drawing:

Figure 1 shows an example of a page containing a scanned photograph, a bar chart generated with a statistical package, and line art generated using spot colors;

Figure 2 is a block diagram showing the architecture of a known color management system;

Figure 3 shows the relationship between an application, the ColorSync Utilities, and the color matching component in the color management system of Figure 2;

Figure 4 diagrams a default matching condition in the color management system of Figure 2, in which input devices match to the system color profile, and output devices match from it to the profile of their device;

Figure 5 is a diagram of a ColorSync™ 1.0 profile record;

Figure 6 illustrates a case in which, in the color management system of Figure 2, matching is performed entirely by a single CMM where both the source and destination profiles have the same `CMMType` and the corresponding CMM component is available;

Figure 7 illustrates the case in which, in the color management system of Figure 2, ColorSync first converts the source data from the source space to XYZ space using the source profile and its corresponding CMM and then converts the

XYZ color space to the destination space using the destination profile and its CMM;

Figure 8 is a functional block diagram of the color management system of the present invention; and

Figure 9 is a diagram of a color profile format in accordance with the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

Referring to Figure 8, the present color management system architecture is divided into five parts: 1) a memory-resident dispatcher, 2) profile management functions, 3) color space conversion functions, 4) color management methods (CMMs), and 5) a standard cross-platform color profile format. The memory-resident dispatcher, profile management functions, color space conversion functions,color management methods and color profiles are identified in Figure 8 by reference numerals 10, 22, 30, 40 and 50, respectively. The profile management, color space conversion, and color management functions are all implemented using the Component Manager 60. Components are dynamically dispatched on demand. In addition, the Component Manager provides database facilities to track each component in the system, allowing the resident dispatcher 10 to call the appropriate component.

The resident dispatcher provides the overall color management administrative framework and manages the interaction between the application, the various color management pieces, and the actual color image. Typically, the dispatcher receives input from the application, analyzes the current profiles and passes the image and profile data to the appropriate CMMs to perform the actual color transformation. Each profile contains information on the preferred CMM for that particular profile. An arbitration scheme within the dispatcher determines which CMM has precedence in the transformation process. In addition to this primary function, the dispatcher calls the appropriate color conversion methods when there exists a mismatch between profile interchange spaces.

On the Macintosh computer system, the color management system consists of an Extension file from which the resident dispatcher is installed at system startup. A separate Control Panel file provides user control of the ColorSync System Profile. The color management system relies on the Component Manager for the basis of the framework which allows plug-and-play capability for third party color-matching implementations.

On 680X0 Macintoshes, the dispatcher is implemented using the Toolbox A-trap mechanism. The interface defines the value for register D0 at the time a function is called. A selector is in the low 16 bits and the parameter size is in the high 16 bits. The functions use pascal stack-based calling conventions. The color management system implements a 680X0 mainline function which internally dispatches individual calls on the selector.

On other computer systems, the specific implementation of the dispatcher will vary.

Device profiles provide color management systems necessary information to convert color data between native device color spaces and device independent color spaces. In the present color management system, color devices are divided into three broad classifications: input devices, display devices and output devices. For each device class, a series of base algorithmic models are described which performs the transformation between color spaces. These models provide a range of color quality and performance results. Each of the base models provides different trade-offs in memory footprint, performance and image quality. The necessary parameter data to implement these models is contained in the required portions of the appropriate device profile descriptions. This required data provides the necessary information for the color management framework default color management module (CMM) to transform color information between native device color spaces.

Selecting an appropriate color matching module to perform a color matching operation is referred to as CMM arbitration. As compared to ColorSync 1.0, the present color management system uses a more efficient arbitration scheme, as follows:

1. If both the source and destination profiles have the same CMMType and the corresponding CMM component is available then matching is performed entirely by that CMM. If the CMM is not available, the color management system uses the default CMM.

2. If either source or destination CMM is not available, that CMM is replaced with the default CMM. If both the source and the destination CMMs are not available, the color management system uses the default CMM and the matching is performed entirely by the default CMM.

3. A concatenated transformation table is built by either the dispatcher of the default CMM sampling the source data space and converting the sampled color points into destination data space using source and destination CMMs. This is done by first converting the source data to the interchange space data using the source CMM, then converting between interchange spaces from source profile to destination profiles, if different, using the color space conversion functions, and finally converting from the interchange data space to the destination data space using the destination CMM.

4. Colors are converted from source to destination data spaces by doing a table look up using the transformation table built in Step 3.

This improved scheme provides the most flexible CMM arbitration by giving the preferred CMM first chance to convert between device space and interchange space, while improving the performance by building a concatenated transformation table. This arbitration scheme also avoids the possible domination of destination CMMs by always answering yes when queried concerning the ability to perform a matching operation.

Referring to Figure 9, the profile structure, rather than being memory-resident, is defined as a series of tagged elements that can be accessed randomly and individually. The collection of tagged elements provides three levels of information: required data, optional data and private data. A tag data header provides a table of contents for the tagging information in each individual profile. The header includes a tag signature and the beginning address offset and size of the tag for each individual tagged element. This

tagging scheme allows the tag data header to be read in, after which the information necessary for a current software application may be randomly accessed and loaded into memory. Since some instances of profiles can be quite large, this arrangement provides significant savings in performance and memory.

The required tags provide the complete set of information necessary for the default CMM to translate color information between the color interchange space and the native device space. A default modeling method for each device class determines which combination of tags is required. For example, a multi-dimensional lookup table is required for output devices, but not for display devices.

A number of optional tags are defined that can be used for enhanced color transformations, including tags that support PostScript Level 2 and calibration, among others.

Private data tags allow CMM developers to add proprietary value to their profiles. Private data tags allows developers to maintain proprietary advantages without sacrificing compatibility.

Referring still to Figure 9, the header provides a set of parameters at the beginning of the profile format. In a preferred embodiment, the header is fixed-length (128 bytes) and contains the following information:

byte(s)	content
0-3	Profile size.
4-7	Identifies the preferred CMM to be used.
8-11	Profile version number.
12-15	Profile/Device class.
16-19	Color space of data (possibly a derived space) [i.e., "the canonical input space"].
20-23	Interchange color space [i.e., "the canonical output space"].
24-35	Date and time this profile was first created.
36-39	'acsp' (0x61637370) profile file signature
40-43	Primary platform target for the profile.
44-47	Flags to indicate various options for the CMM such as distributed processing and caching options.
48-51	Device manufacturer of the device for which this profile is created.
52-55	Device model of the device for which this profile is created.
56-63	Device attributes unique to the particular device setup such as media type.
64-67	Specifies the rendering intent of this profile for the CMM. Photographic, spot colors and business graphics are the three intents required to be supported with default values of 0, 1 and 2 respectively.
68-79	The relative XYZ values of the white point of the color space.
80-91	The relative XYZ values of the black point of the color space.
92-127	36 bytes reserved for future expansion.

In a preferred embodiment, the profile format supports a variety of both device-dependent and device independent color spaces divided into three basic families: 1) CIEXYZ based, 2) RGB based, and 3) CMY based. A subset of the

CIEXYZ based spaces are also defined as exchange spaces. Other device dependent color spaces may be also used. Any of the defined color spaces may be used as a device space, which corresponds to the source or destination color space in a color modeling session.

A key component of a color profile is the choice of color interchange spaces. These spaces provide the "universal language" with which any number of color devices can communicate color information with each other. Most color interchange spaces today are based on the CIE 1931 standard observer. This experimentally derived standard observer provides a very good representation of the human visual system's color matching capabilities. Two color spaces based on these results have become very popular in color management systems: the 1931 CIEXYZ space and the 1976 CIELAB space. The CIEXYZ space represents a linear transformation of the derived matching responses and the CIELAB space represents a transformation of the CIEXYZ space into one that is nearly perceptually uniform. This uniformness allows color errors to be equally weighted throughout its domain.

Three color interchange space encodings are supported in the preferred embodiment 16 bit per component CIEXYZ, 16 bit per component CIELAB and 8 bit per component CIELAB. While supporting multiple interchange spaces increases the complexity of color management, it provides immense flexibility in addressing different user requirements such as color accuracy and memory footprint.

The device profiles along with the appropriate device models provide a translation mechanism between the native device color space and one of these three color interchange space encodings.

The choice of device space and color interchange space is indicated within the profile itself as follows:

Color Space	Signature
XYZData	'XYZ'
labData (16 bit per component)	'Lab'
lab8Data (8 bit per component)	'Lab8'
luvData	'Luv'
YxyData	'Yxy'
rgbData	'RGB'
grayData	'GRAY'
hsvData	'HSV'
hlsData	'HLS'
cmykData	'CMYK'
cmyData	'CMY'

Requiring tags with profiles is a way to provide a common base level of functionality. If a custom CMM is not present, then the default CMM will have enough information to adequately model the data. The required data for different devices and the particular models implied by the required data are set forth in Appendix 1 below. While the required data might not provide the highest level of quality obtainable with optional data and private data, the data provided is adequate for sophisticated device modeling.

Profiles need to be validated to contain required color information for the tagged CMM to use. A `ValidateProfile` function is provided to test whether a specified profile contains the minimum set of elements. This function is dispatched to the CMM specified by the `Profile CMMTypeTag` element. If the CMM is not available then the default Apple CMM is called. Only the existence of `Profile` elements is checked. The element content and size are not.

The dispatcher, upon receiving a request to validate a profile, will do the following steps.

1. Get the name of the preferred CMM from the profile.
2. If available, dispatch the preferred CMM to do the validation.
3. If not available, dispatch the default CMM to do the validation.

Whether or not a tag is required depends on the profile type. Tags that are required in some profile types may be optional in other profile type. In addition, optional tags not generally required in a profile type include the following:

Tag Name	General Description
calibrationDateTimeTag	Profile calibration date and time
gcrTag	Gray component replacement curve
ucrTag	Under color removal curve
charTargetTag	Characterization target such as IT8/7.2
luminanceTag	Absolute luminance for emissive device
measurementTag	Alternative measurement specification such as a D65 illuminant instead of the default D50
viewingConditionsTag	Specifies viewing condition parameters
namedColorTag	Named color dictionary for converting between named color sets and interchange spaces
preview0Tag	Preview transformation: 8 or 16 bit data: intent value of 0
preview1Tag	Preview transformation: 8 or 16 bit data: intent value of 1
preview2Tag	Preview transformation: 8 or 16 bit data: intent value of 2
monitoringTag	monitoring attributes such as frequency, angle and spot
deviceMfgDescTag	displayable description of device manufacturer
deviceModelDescTag	displayable description of device model

technologyTag	Device technology information such as LCD, CRT, Dye Sublimation, etc.
srcDeviceAttributesTag	Device attributes for source profile
srcDeviceModelTag	Device model designation for source profile
srcManufacturerTag	Device manufacturer for source profile
srcDeviceMfgDescTag	Displayable description of source device manufacturer
srcDeviceModelDescTag	Displayable description of source device model
srcTechnologyTag	Source device technology information such as LCD,CRT, Dye Sublimation, etc.
ps2CRDTag	PostScript Level 2 color rendering dictionary
ps2CSATag	PostScript Level 2 color space array
ps2RenderingIntentTag	PostScript Level 2 Rendering Intent

The latter three optional tags (PostScript Level 2 color rendering dictionary, PostScript Level 2 color space array, and PostScript Level 2 Rendering Intent, described in detail in the *PostScript Language Reference Manual*, Second Edition, provide all the necessary information to produce color-matched output on a PostScript device. Since these tags are optional, however, the present color management system provides facilities to derive equivalent information from color profiles that do not include them. The conversion between color profiles and PostScript is built into the color management system, utilizing its modular framework and code resource database manager (i.e., the Component Manager).

The color management system, by using the code resource database manager, can dynamically dispatch the preferred code module to generate the equivalent PostScript for the given color profile. The preferred color conversion code module is tagged in the profile.

If the preferred color conversion code module is not available, the default code module will be invoked to convert the profile into PostScript. In this case, only the public profile information is used to generate PostScript.

There are three functions defined in the color management system API for requesting conversion of profile data to PostScript. They are:

•`GetPS2ColorSpace` - returns 7 bit ASCII or 8 bit binary characters usable as the parameter to the PostScript `setColorSpace` instruction.

•`GetPS2ColorRenderingIntent` - returns 7 bit ASCII or 8 bit binary characters usable as the parameter to the PostScript `setRenderingIntent` instruction.

•`GetPS2ColorRendering` - returns 7 bit ASCII or 8 bit binary characters usable as the parameter to the Postscript `setColorRendering` instruction.

To convert a color profile into PostScript, the driver or application invokes one of the PostScript functions through the API provided by the color management system. The following steps describe the dispatching algorithm:

color management dispatcher checks the preferred CMM name tagged with profile

check with code module data base manager if the preferred module is available

if the preferred module is found
dispatch the code module with the Postscript API functions

else

dispatch the default code module with the Postscript API functions

When a CMM is dispatched with the PostScript APIs, it will go through the following steps to pass the PostScript equivalent back to the caller:

validate the color profile
allocate data buffer for transfer

check if there is PostScript tag included in the profile

```
if found the right tag
    convert the encoding format as requested, if necessary
        stream out the Postscript using the data buffer and the caller
supplied call back function

    else

        generate the PostScript equivalent with the requested encoding
format

        stream out the Postscript using the data buffer and the caller
supplied call back function.
```

A matching session typically is set up between two device profiles, a source profile and a destination profile. A typical scenario is a graphic designer who has created a drawing on a computer monitor using a drawing application. The designer will often want to print the design out using a color printer so the customer can see it. To ensure color fidelity, a matching session is set up with the source profile being the display profile for the computer monitor and the destination profile being the color printer profile.

However, since different color devices have different color gamuts, some colors on one device may not be reproducible on another. In this case, a color close to the target will be selected. If the accuracy of the color output is critical, the designer may want to do a "preview" of the color output on the computer monitor. A matching session with three or more profiles is required in order to convert color from display space to printer space and then convert back to display space for preview purposes. The process of setting up a multiple profile matching session and reducing the transformations down to one is called profile concatenation.

To set up a multiple profile matching session, there are two methods that the dispatcher can use.

Method 1:

1. Through the color management system application program interface (API), an application sets up profile concatenation by providing a list of profiles with a "key profile" which indicates the preferred CMM to do the operation.

2. If the key CMM is not available, the default CMM is dispatched to do the concatenation.

3. If the key CMM is available, it will be dispatched to do the concatenation. If the key CMM returns an error, then the default CMM is used.

The second method can be employed by the dispatcher or by the CMMs to perform profile concatenation.

Method 2:

1. Through the color management system API, the application sets up profile concatenation by providing a list of profiles.

2. The dispatcher checks if the preferred CMM for each profile in the list is available. If not, it is replaced with the default CMM.

3. A concatenated transformation table is built by sampling the source data space and converting the sampled color points into destination data space using the profiles in the list and CMMs associated with each profile. This is done by the following steps:

a. Convert the source data to the interchange space data using the source CMM.

b. Convert between interchange spaces from source profile to the next profile, if different, using the color space conversion functions.

c. Convert from the interchange space to the device color space using the designated CMM.

d. Convert from the device color space to the interchange space using the designated CMM.

e. Repeat steps b to d until all the profiles in the list except the last

one is exhausted.

f. Convert from the interchange space to the destination data space using the destination CMM.

4. Colors are converted from source to destination data spaces by doing a table look up using the transformation table built in Step 3.

The concatenated profile can be stored as a special type of profile (called a device link profile). If the user establishes a workflow pattern of designing and proofing documents, a device link profile can be saved to reflect the color conversion sequence by the user's workflow.

Creating a device link profile is very similar to the profile concatenation process. Two algorithms can be used by the dispatcher. The only difference from profile concatenation is that in step 4, instead of converting colors from source data space to destination data space, a device link profile is simply created with the transformation table stored in it.

In a preferred embodiment, a device link profile is one of three special profiles, or transformations, in addition to the three basic device profile classes (input, display and output). These transformations provide a standard implementation for use by the CMM in general color processing or for the convenience of CMMs which may use these types to store calculated transforms. These three transformation classes are: device link, color space conversion, and abstract transformations. Device link transformations, as previously described, provide a mechanism in which to save and store a series of device profiles and color transformations in a concatenated format. This is extremely useful for workflow issues where a combination of devices and transforms are used repeatedly.

Color space conversion transformations are used as a convenient method for CMMs to convert between different color spaces. This is particularly useful if two device profiles support different interchange spaces. Conversions

between base color spaces and derived color spaces are performed with known equations. The conversion functions are implemented in a stand-alone Component, making the routines available to the default CMM and to third-party CMMs.

Finally, abstract color transformations provide a generic method for users to make subjective color changes to images or graphic objects. This transformation is encapsulated within a three dimensional table structure to provide maximal flexibility. These transforms, set forth in detail in Appendix 2, have the following signatures:

`'link'`	device link transformations,
`'spac'`	color space conversion transformations,
`'abst'`	abstract transformations.

The external API implemented by the dispatcher is set forth in Appendix 3.

It will be appreciated by those of ordinary skill in the art that the invention can be embodied in other specific forms without departing from the spirit or essential character thereof. The presently disclosed embodiments are therefore considered in all respects to be illustrative and not restrictive. The scope of the invention is indicated by the appended claims rather than the foregoing description, and all changes which come within the meaning and range of equivalents thereof are intended to be embraced therein.

APPENDIX 1

1.1 Input Profile

This profile represents input devices such as scanners and digital cameras.

1.1.1 Monochrome Input Profiles

Tag Name	General Description
profileDescriptionTag	Structure containing invariant and localizable versions of the profile name for display
grayTRCTag	Gray tone reproduction curve

The mathematical model implied by this data is:

$$interchange = grayTRC[device]$$

This represents a simple tone reproduction curve adequate for most monochrome input devices. The interchange values in this equation should represent the achromatic channel of the interchange space. If the inverse of this is desired, then the following equation is used:

$$device = grayTRC^{-1}[interchange]$$

1.1.2 RGB Input Profiles

Tag Name	General Description
profileDescriptionTag	Structure containing invariant and localizable versions of the profile name for display
redColorantTag	Red colorant XYZ relative tristimulus values
greenColorantTag	Green colorant XYZ relative tristimulus values
blueColorantTag	Blue colorant XYZ relative tristimulus values
redTRCTag	Red channel tone reproduction curve
greenTRCTag	Green channel tone reproduction curve
blueTRCTag	Blue channel tone reproduction curve

The forward mathematical model implied by this data is :

$$linear_R = redTRC[device_r]$$
$$linear_G = greenTRC[device_g]$$
$$linear_B = blueTRC[device_b]$$
$$\begin{vmatrix} interchange_X \\ interchange_Y \\ interchange_Z \end{vmatrix} = \begin{vmatrix} redColorant_X & greenColorant_X & blueColorant_X \\ redColorant_Y & greenColorant_Y & blueColorant_Y \\ redColorant_Z & greenColorant_Z & blueColorant_Z \end{vmatrix} \begin{vmatrix} linear_R \\ linear_G \\ linear_B \end{vmatrix}$$

This represents a simple linearization followed by a linear mixing model. The three tone reproduction curves linearize the raw values with respect to the luminance (Y) dimension of the CIEXYZ interchange space. The 3x3 matrix converts these linearized values into XYZ values for the CIEXYZ interchange space. The inverse model is given by the following equation,

$$\begin{vmatrix} linear_R \\ linear_G \\ linear_B \end{vmatrix} = \begin{vmatrix} redColorant_X & greenColorant_X & blueColorant_X \\ redColorant_Y & greenColorant_Y & blueColorant_Y \\ redColorant_Z & greenColorant_Z & blueColorant_Z \end{vmatrix}^{-1} \begin{vmatrix} interchange_X \\ interchange_Y \\ interchange_Z \end{vmatrix}$$
$$device_r = redTRC^{-1}[linear_R]$$
$$device_g = greenTRC^{-1}[linear_G]$$
$$device_b = blueTRC^{-1}[linear_B]$$

1.1.3 CMYK Input Profiles

Tag Name	General Description
profileDescriptionTag	Structure containing invariant and localizable versions of the profile name for display
AToB0Tag	Device to Interchange space : 8 or 16 bit data

The AToB0Tag represents a device model described by the Lut8Type or Lut16Types. This tag provides the parameter data for an algorithm that includes a set of non-interdependent per-channel tone reproduction curves, a three dimensional lookup table and a set of non-interdependent per-channel linearization curves.

1.2 Display Profile

This profile represents display devices such as monitors.

1.2.1 Monochrome Display Profiles

Tag Name	General Description
profileDescriptionTag	Structure containing invariant and localizable versions of the profile name for display
grayTRCTag	Gray tone reproduction curve

The tone reproduction curve provides the necessary information to convert between a single device channel and the CIEXYZ interchange space.

The mathematical model implied by this data is:

$$interchange = grayTRC[device]$$

This represents a simple tone reproduction curve adequate for most monochrome input devices. The interchange values in this equation should represent the achromatic channel of the interchange space. If the inverse of this is desired, then the following equation is used:

$$device = grayTRC^{-1}[interchange]$$

1.2.2 RGB Display Profiles

Tag Name	General Description
profileDescriptionTag	Structure containing invariant and localizable versions of the profile name for display
redColorantTag	Relative XYZ values of red phosphor
greenColorantTag	Relative XYZ values of green phosphor
blueColorantTag	Relative XYZ values of blue phosphor
redTRCTag	Red channel tone reproduction curve
greenTRCTag	Green channel tone reproduction curve
blueTRCTag	Blue channel tone reproduction curve

This model is based on a three non-interdependent per-channel tone reproduction curves to convert between linear and non-linear rgb values and a 3x3 matrix to convert between linear rgb values and relative XYZ values. The mathematical model implied by this data is :

$$linear_R = redTRC[in_r]$$
$$linear_G = greenTRC[in_g]$$
$$linear_B = blueTRC[in_b]$$
$$\begin{vmatrix} X \\ Y \\ Z \end{vmatrix} = \begin{vmatrix} redColorant_X & greenColorant_X & blueColorant_X \\ redColorant_Y & greenColorant_Y & blueColorant_Y \\ redColorant_Z & greenColorant_Z & blueColorant_Z \end{vmatrix} \begin{vmatrix} linear_R \\ linear_G \\ linear_B \end{vmatrix}$$

This represents a simple linearization followed by a linear mixing model. The three tone reproduction curves linearize the raw values with respect to the luminance (Y) dimension of the CIEXYZ interchange space. The 3x3 matrix converts these linearized values into XYZ values for the CIEXYZ interchange space. The inverse model is given by the following equation,

$$\begin{vmatrix} linear_R \\ linear_G \\ linear_B \end{vmatrix} = \begin{vmatrix} redColorant_X & greenColorant_X & blueColorant_X \\ redColorant_Y & greenColorant_Y & blueColorant_Y \\ redColorant_Z & greenColorant_Z & blueColorant_Z \end{vmatrix}^{-1} \begin{vmatrix} interchange_X \\ interchange_Y \\ interchange_Z \end{vmatrix}$$
$$device_r = redTRC^{-1}[linear_R]$$
$$device_g = greenTRC^{-1}[linear_G]$$
$$device_b = blueTRC^{-1}[linear_B]$$

1.3 Output Profile

This profile represents output devices such as printers and film recorders. The LUT tags that are required by the printer profiles contain either the 8 bit or the 16 bit LUTs exclusively as described in the LUT tags. The bit precision supported must be consistent for all of the LUT tags. The LUT algorithm for interchange space to device space transformations process data sequentially through a matrix, input tables, a color LUT, and output tables.

1.3.1 Monochrome Output Profiles

Tag Name	General Description
profileDescriptionTag	Structure containing invariant and localizable versions of the profile name for display
grayTRCTag	Gray tone reproduction curve

The tone reproduction curve provides the necessary information to convert between

a single device channel and the CIEXYZ interchange space.

The mathematical model implied by this data is:

$$interchange = grayTRC[device]$$

This represents a simple tone reproduction curve adequate for most monochrome input devices. The interchange values in this equation should represent the achromatic channel of the interchange space. If the inverse of this is desired, then the following equation is used:

$$device = grayTRC^{-1}[interchange]$$

1.3.2 RGB and CMYK Output Profiles

Tag Name	General Description
profileDescriptionTag	Structure containing invariant and localizable versions of theprofile name for display
AToB0Tag	Device to Interchange space : 8 or 16 bit data: intent of 0
BToA0Tag	Interchange to Device space : 8 or 16 bit data: intent of 0
gamut0Tag	Out of Gamut : 8 or 16 bit data: intent of 0
AToB1Tag	Device to Interchange space : 8 or 16 bit data: intent of 1
BToA1Tag	Interchange to Device space : 8 or 16 bit data: intent of 1
gamut1Tag	Out of Gamut tag : 8 or 16 bit data: intent of 1
AToB2Tag	Device to Interchange space : 8 or 16 bit data: intent of 2
BToA2Tag	Interchange to Device space : 8 or 16 bit data: intent of 2
gamut2Tag	Out of Gamut tag : 8 or 16 bit data: intent of 2

These tags represent a device model described by the Lut8Type or Lut16Types. The intent values described in these tags directly correlate to the value of the rendering intent header flag of the source profile in the color modeling session (0 = photographic, 1 = logo color, and 2 = business graphics). In essence, each of these tags provides the parameter data for an algorithm that includes a 3x3 matrix, a set of non-interdependent per-channel tone reproduction curves, a three dimensional lookup table and a set of non-interdependent per-channel linearization curves. The algorithmic details of this model and the intent of each tag is given in the following sub-section, specifying the general lookup table tag element structure for 16-bit data.

1.4 lut16Type

This structure converts an 16 bit input color into an 16 bit output color. This type contains four processing elements: a 3 by 3 matrix (only used when the input color space has three components), a set of one dimensional input lookup tables, a multidimensional lookup table, and a set of one dimensional output tables. Data is pro-

cessed using these elements via the following sequence:
(matrix) -> (1d input tables) -> (multidimensional lookup table) -> (1d output tables).

byte(s)	content
0-3	'mft2' (0x6D667432) type descriptor
4-7	reserved, must be set to 0
8	Number of Input Channels
9	Number of Output Channels
10	Number of CLUT grid points (identical for each side)
11	Reserved for padding (required to be 0x00)
12-15	Encoded e00 parameter
16-19	Encoded e01 parameter
20-23	Encoded e02 parameter
24-27	Encoded e10 parameter
28-31	Encoded e11 parameter
32-35	Encoded e12 parameter
36-39	Encoded e20 parameter
40-43	Encoded e21 parameter
44-45	Encoded e22 parameter
46-47	Number of input table entries
48-49	Number of output table entries
50-n	input table values
n+1-m	CLUT values
m+1-o	output table values

The input, output and CLUT tables are arrays of 16 bit unsigned values. Each input table entry is a two byte integer and each table has up to 4096 entries. Each input table entry is appropriately normalized to the range 0-65535. The inputTable is of size InputChannels * inputTableEntries * 2 bytes.

The matrix is organized as an 3 by 3 array. The dimension corresponding to the matrix rows varies least rapidly and the dimension corresponding to the matrix columns varies most rapidly and is shown in matrix form below. Each matrix entry is a four byte number with one sign bit, 15 integer bits, and 16 fractional bits.

$$\begin{vmatrix} e00 & e01 & e02 \\ e10 & e11 & e12 \\ e20 & e21 & e22 \end{vmatrix}$$

When using the matrix of an output profile, and the input data is XYZ, we have

$$\begin{vmatrix} X' \\ Y' \\ Z' \end{vmatrix} = \begin{vmatrix} XX & XY & XZ \\ YX & YY & YZ \\ ZX & ZY & ZZ \end{vmatrix} \begin{vmatrix} X \\ Y \\ Z \end{vmatrix}$$

Each input X, Y or Z is an unsigned 1.15 number and each matrix entry is a signed 15.16 number. Therefore, each multiplication in the matrix multiply is 1.15 * s15.16 = s16.31 and the final sum is also s16.31. From this sum we take bits 31-16 as the unsigned integer result for X', Y', or Z'. These are then used as the inputs to the input tables of the multidimensional LUT. This normalization is used since the number of fractional bits in the input data must be maintained by the matrix operation.

The matrix is mandated to be an identity matrix for input and display profiles. In addition, the matrix is mandated to be an identity matrix for output profiles when the interchange space is CIELAB.

Each CLUT is organized as an n-dimensional array with a variable number of grid points in each dimension, where n is the number of input channels(input tables) in the transform. The dimension corresponding to the first input channel varies least rapidly and the dimension corresponding to the last input channel varies most rapidly. Each grid point value contains m two byte integers, where m is the number of output functions. The first sequential two byte integer of the entry contains the function value for the first output function, the second sequential two byte integer of the entry contains the function value for the second output function, and so on until all the output functions have been supplied. The equation for computing the size of the CLUT is :

$$CLUTsize = LUTDimensions^{InputChannels} \bullet OutputChannels \bullet 2bytes$$

The CLUT data must be padded to a 4 byte boundary with zeros at the end of the data if necessary.

Each output table is a two byte integer and has up to 4096 entries. The outputTable is of size OutputChannels * outputTableEntries * 2 bytes.

When using this type, it is necessary to assign each color space component

to an input and output channel. The following table shows these assignments. The channels are numbered according to the order in which their table occurs. Note that additional color spaces can be added simply by defining the signature, channel assignments, and creating the tables.

Color Space	Channel 1	Channel 2	Channel 3	Channel 4
'XYZ'	X	Y	Z	
'Lab'	L	a	b	
'Luv'	L	u	v	
'Yxy'	Y	x	y	
'RGB'	R	G	B	
'GRAY'	K			
'HSV'	H	S	V	
'HLS'	H	L	S	
'CMYK'	C	M	Y	K
'CMY'	C	M	Y	

APPENDIX 2

2.1 DeviceLink Transform

This transform represents a one-way link or connection between devices. It does not represent any device model nor can it be embedded into images.

Tag Name	General Description
profileDescriptionTag	Structure containing invariant and localizable versions of the profile name for display
AToB0Tag	Actual transformation parameter structure (this is an exclusive or) 8 or 16 bit data
srcDeviceAttributesTag	Device attributes for source profile
srcBlackPointTag	The relative XYZ values of the black point of the source device
srcDeviceModelTag	Device model designation for source profile
srcManufacturerTag	Device manufacturer for source profile
srcWhitePointTag	The relative XYZ values of the white point of the source device.

The AToB0Tag represents a device model described by the Lut8Type or Lut16Types. This tag provides the parameter data for an algorithm that includes a 3x3 matrix, a set of non-interdependent per-channel tone reproduction curves, a three dimensional lookup table and a set of non-interdependent per-channel linearization curves.

2.2 ColorSpaceConversion Transform

This transform provides the relevant information to perform a color space transformation between the interchange color spaces. It does not represent any device model nor can it be embedded into images.

Tag Name	General Description
profileDescriptionTag	Structure containing invariant and localizable versions of the profile name for display
BToA0Tag	Inverse transformation parameter structure (this is an exclusive or) 8 or 16 bit data
AToB0Tag	Actual transformation parameter structure (this is an exclusive or) 8 or 16 bit data

srcBlackPointTag	The relative XYZ values of the black point of the source space.
srcWhitePointTag	The relative XYZ values of the white point of the source space.

The AToB0Tag and BToA0Tag represent a device model described by the Lut8Type or Lut16Types. This tag provides the parameter data for an algorithm that includes a 3x3 matrix, a set of non-interdependent per-channel tone reproduction curves, a three dimensional lookup table and a set of non-interdependent per-channel linearization curves.

2.3 Abstract Transform

This transform represents abstract transforms and does not represent any device model nor can it be embedded into images. Color transformations using abstract profiles are performed within a single interchange space.

Tag Name	General Description
profileDescriptionTag	Structure containing invariant and localizable versions of the profile name for display
AToB0Tag	Actual transformation parameter structure (this is an exclusive or) 8 or 16 bit data

The AToB0Tag represents a device model described by the Lut8Type or Lut16Types. This tag provides the parameter data for an algorithm that includes a 3x3 matrix, a set of non-interdependent per-channel tone reproduction curves, a three dimensional lookup table and a set of non-interdependent per-channel linearization curves.

APPENDIX 3

Profile File and Element Access

OpenProfileFile

Open specified `CMProfile` file. Return a `Profile` reference. The profile file is opened with shared read/write permission.

```
pascal
CMError
OpenProfileFile(Profile *prof,
                const FSSpec *fileSpec);
```

UpdateProfileFile

Rewrite disk file if elements of specified `Profile` have been added or changed. An error will be returned if the profile file is open by another program.

```
pascal
CMError
UpdateProfileFile(Profile prof)
```

CloseProfile

Close data file and free memory allocations associated with specified `Profile`. Changes to the file are not automatically recorded. See `UpdateProfileFile(...)`.

```
pascal
CMError
CloseProfile(Profile prof)
```

NewProfile

Create new empty `Profile` and associated backing file. The `Profile` must be populated and updated by the client program. The default disk file type is 'prof'. The default profile contents include the `CM2Header` and an empty element table. All fields of the `CM2Header` will be set to zeroes except `size` and `profileVersion`.

```
pascal
CMError
NewProfile(Profile *prof, const FSSpec *backingFileSpec)
```

CopyProfile

Duplicate a `Profile`. Temporary changes which have been made to the source `Profile` are included in target `CMProfile` disk file.

```
pascal
CMError
```

```
CopyProfile(Profile *profCopy,
            const FSSpec *copyFileSpec,
            Profile prof);
```

ValidateProfile

Test whether Profile contains the minimum set of elements. This function is dispatched to the CMM specified by the Profile CMMTypeTag element.

```
pascal
CMError
ValidateProfile(Profile prof, Boolean *valid);
```

FlattenProfile

Flatten specified Profile to the external CMProfile format. A caller supplied FlattenProcUPP is called to perform the actual data transfer. The flags parameter specifies options for the content of the flattened CMProfile.

This function is used to create an embedded CMProfile which can be used for graphic documents. The element subset is normally specified by the sourceProfElements flag. The qd32KLimit flag should be used for Quickdraw PicComments.

This routine is dispatched to the CMM Component specified by the Profile CMMTypeTag element. If the CMM is not available then the default Apple CMM is called. The default elements are flattened and the function result is CMMethodNotFound.

```
enum {
   AtoBElements=  1,
   BtoAElements=  2,
   intent0      =  4,
   intent1      =  8,
   intent2      = 10,
   qd32KLimit   = 20
};
```

AtoBElements Include elements necessary for the source side of a matching session. Common case for a graphic document profile tag.

BtoAElements Include elements necessary for the destination side of a matching session.

intent0 Include elements necessary for rendering intent 0.

intent1 Include elements necessary for rendering intent 1.

intent2 Include elements necessary for rendering intent 2.

qd32KLimit Flattened CMProfile may not exceed 32K, the limit for QuickDraw PicComments.

```
pascal
CMError
FlattenProfile(Profile prof, long flags, FlattenProcUPP
proc,
                void *refCon);
```

Parameters

prof (in) Source `Profile`

flags (in) Specifies options for destination`CMProfile`.

proc (in) Function for data transfer

refCon (in) Client data which is passed as a parameter to calls to`proc`.

Result codes

noErr

paramErr Result specified by flags is not possible.
i.e. sourceProfileElements |
destProfileElements |
resolveReferences |
qd32KLimit

`CMMethodNotFound`CMM Corresponding to `Profile` not available

System or ColorSync result code if an error occurs

UnflattenProfile

Unflatten an external format CMProfile to an independent disk file. A client-supplied FlattenProcUPP is called to perform the actual data transfer.

This function may be used to create a Profile reference from a CMProfile embedded in a graphic document. The caller is responsible for calling OpenProfileFile(...) and then clean-up by calling CloseProfile(...) and FSpDelete(...) when use of the profile is completed.

This routine is dispatched to the CMM Component specified by the Profile CMMTypeTag element. If the CMM is not available then the default Apple CMM is called. The function result in this case will be CMMethodNotFound.

```
pascal
CMError
UnflattenProfile(FSSpec *resultFileSpec, FlattenProcUPP
proc,
                 void *refCon);
```

Parameters

resultFileSpec (out) Returns profile file spec if function result is noErr or CMMethodNotFound. Otherwise undefined.

proc (in) Function for data transfer

refCon (in) Client data which is passed as a parameter to calls to proc.

Result codes

noErr

CMMethodNotFound CMM corresponding to Profile not available

System or ColorSync result code if an error occurs

`FlattenProfile(...)` and `UnflattenProfile(...)` use a caller-supplied `flattenProc` function to perform the actual data transfer. This `flattenProc` function is also a parameter to the PostScript-related utility functions. This mechanism allows the caller to control the low level transfer.

Communication with the `flattenProc` function uses a `command` parameter.

```
enum {
    OpenReadSpool = 1,
    OpenWriteSpool,
    ReadSpool,
    WriteSpool,
    CloseSpool
};
```

OpenReadSpool Begin the process of readng data.
OpenWriteSpoolBegin the process of writing data.
ReadSpool Read size bytes.
WriteSpool Write size bytes.
CloseSpool Finish data transfer.

```
typedef pascal OSErr (*FlattenProcPtr) (unsigned char com-
mand,
                                        unsigned long
*size,
                                        void *data, void
*refCon);
#if USESROUTINEDESCRIPTORS
typedef UniversalProcPtrFlattenProcUPP;
#else
typedef FlattenProcPtrFlattenProcUPP;
#endif
```

Parameters
command(in) `Specifies operation.`

size (in/out)`Specifies data size on input.`
Returns actual size transferred.

data (in) Address of buffer.

refCon (in) Client data which is passed as a parameter to calls to the `flattenProc`.

ProfileElementExists

Test whether an element with specified tag signature is present in `Profile`.

```
pascal
CMError
ProfileElementExists(Profile prof, OSType tag,
                     Boolean *found);
```

CountProfileElements

Return one-based count of elements contained in `Profile`. Tags which are references are counted as elements.

```
pascal
CMError
CountProfileElements(Profile prof, long *elementCount)
```

GetProfileElement

Get element data for specified tag. The caller is responsible for allocating storage.

```
pascal
CMError
GetProfileElement(Profile prof, OSType tag,
                  long *elementSize, void *elementData);
```

GetProfileHeader

Get copy of header for the specified `Profile`.

```
pascal
CMError
GetProfileHeader(Profile prof, AppleProfileHeader *head-
er);
```

GetPartialProfileElement

Get part of element data for specified tag. The caller is responsible for allocating storage.

```
pascal
CMError
GetPartialProfileElement(Profile prof, OSType tag,
                         unsigned long offset,
                         unsigned long byteCount,
                         void *elementData);
```

SetProfileElementSize

Reserve size of element data for specified tag. This function must be used before calling `SetPartialProfileElement(…)`.

```
pascal
CMError
SetProfileElementSize(Profile prof, OSType tag,
                      unsigned long elementSize);
```

SetPartialProfileElement

Set part of element data for specified tag.

```
pascal
CMError
SetPartialProfileElement(Profile prof, OSType tag,
                         unsigned long offset,
                         unsigned long byteCount,
                         void *elementData);
```

GetIndeProfileElementInfo

Obtain element tag and data size by index in the range of 1..elementCount (as returned by `CountProfileElements(…)`).

```
pascal
CMError
GetIndProfileElementInfo(Profile prof, unsigned long in-
dex,
                         OSType *tag, unsigned long *ele-
mentSize,
                         Boolean *refs);
```

GetIndProfileElement

Get data for element at specified index. Caller is responsible for allocating storage.

```
pascal
CMError
GetIndProfileElement(Profile prof, unsigned long index,
                     unsigned long *elementSize, void *el-
ementData);
```

SetProfileElement

Set element data for specified tag. If an element with the specified tag is already present in the `Profile`, the existing element data is replaced.

```
pascal
CMError
SetProfileElement(Profile prof, OSType tag,
```

```
                    unsigned long elementSize, void *ele-
mentData);
```

SetProfileHeader

Set header for the specified `Profile`.

```
pascal
CMError
SetProfileHeader(Profile prof, const AppleProfileHeader
*header);
```

SetProfileElementReference

Add tag to `Profile` which references data corresponding to a previously set element.

After successful completion of this function there will be more than one tag corresponding to a single piece of data. All of these tags are "equal citizens". If `SetProfileElement(_)` is subsequently called for one of the common tags then the new element data will be "appended" to the `rofile` rather than replacing the existing data.

It is possible to set a reference to an element which was originally a reference itself without circularity.

```
pascal
CMError
SetProfileElementReference(Profile prof, OSType element-
Tag,
                           OSType referenceTag)
```

RemoveProfileElement

Remove element with specified tag signature from `Profile`.

```
pascal
CMError
RemoveProfileElement(Profile prof, OSType tag);
```

Low-level Matching

To use the low level routines you first create a color matching world, or CWorld, which establishes how matching will take place between the given source and destination profiles. Calling NewColorWorld gives you a color matching world and DisposeColorWorld gets rid of it.

You can match individual colors by passing the color and the color matching world to the matching utilities. This allows you to set up your matching parameters one time, do all the matching you want using the color matching world, and then clean up. Furthermore, it allows you to send and receive colors directly in XYZ, CMYK, or Gray color spaces rather than just RGB.

You can also match an individual pixel map, or determine whether or not a list of colors is in the gamut of a particular color space.

NCWNewColorWorld

Set up a matching session for `src` and `dst` `Profiles`. The CMM Component(s) involved will fetch the `Profile` elements necessary for the matching session.

The `src` and `dst` `Profiles` may be closed with `CloseProfile(...)` after calling this function.

```
pascal
CMError
NCWNewColorWorld(CWorld *myCWorld, Profile src, Profile
dst);
```

NCWConcatColorWorld

Set up a session for color processing which includes a set of `Profiles`. The CMM Component specified by the `Profile` at `ConcatProfileSet.keyIndex` will fetch the `Profile` elements necessary for the session.

The specified `Profiles` may be closed with `CloseProfile(...)` after calling this function.

```
pascal
CMError
NCWConcatColorWorld(CWorld *myCWorld,
                    ConcatProfileSet *profileSet);
```

Parameters

myCWorld(in/out) Returns reference to session if function result is `noErr`. Otherwise undefined.

profileSet(in) `ConcatProfileSet` contains an array of `Profiles` which describe the processing to be carried out.

The `profileSet array` is in processing order— Source through Destination. A minimum of one `Profile` is required.

The `CMMType` element of `Profile` at `keyIndex` specifies the CMM which is entirely responsible for the session.

```
struct ConcatProfileSet
{
  unsigned shortkeyIndex;/* zero-
```

```
based */
                                unsigned shortcount;
                                Profile   profileSet[1];
                              };
```

Result codes
noErr

System or ColorSync result code if an error occurs.

CWNewLinkProfile

This function allows the creation of a new Profile which embodies the transforms included by NCWConcatColorWorld(...).

```
pascal
CMError
CWNewLinkProfile(Profile *prof,
                 const FSSpec *backingFileSpec,
                 ConcatProfileSet *profileSet)
```

Parameters
myCWorld(in/out)Returns reference to session if function result is noErr. Otherwise undefined.

backingFileSpec (in)File spec for the resulting CMProfile.

profileSet(in) ConcatProfileSet contains an array of Profiles which describe the processing to be carried out.

The profileSet array is in processing order— Source through Destination. A minimum of one Profile is required.

The CMMType element of Profile at keyIndex specifies the CMM which is entirely responsible for the session.

Result codes
noErr

System or ColorSync result code if an error occurs.

CWDisposeColorWorld

Free private storage associated with a CWorld.

```
pascal
```

```
void
CWDisposeColorWorld(CWorld myCWorld);
```

CWMatchColors

Match colors according to the `Profiles` corresponding to a `CWorld`. On entry, the `CMColor` values are taken to be the `dataType` of the Source `Profile`. On exit, the `CMColor` values are transformed to the `dataType` of the Destination `Profile`.

Matching sessions set up with one of the Multichannel color data types are supported with this function.

```
pascal
CMError
CWMatchColors(CWorld myCWorld, CMColorList myColors,
              long count);
```

CWCheckColors

Gamut test colors according to the `Profiles` corresponding to `myCWorld`. On entry, the `CMColor` values are taken to be the `dataType` of the Source `Profile`. The gamut test is a preflight of color-matching with this `CWorld`. The result bit array indicates whether the colors in the list are in or out of gamut for the Destination `Profile`.

Matching sessions set up with one of the Multichannel color data types are supported with this function.

```
pascal
CMError
CWCheckColors(CWorld myCWorld, CMColorList myColors,
              unsigned long count, CMGamutResult result);
```

CWMatchBitMap

Match pixel data of `bitMap` according to the `Profiles` corresponding to `myCWorld`.

```
typedef pascal Boolean (*BitMapCallBackProcPtr)(long
progress,
                                                void *ref-
Con);

#if USESROUTINEDESCRIPTORS
typedef UniversalProcPtr BitMapCallBackUPP;
#else
typedef BitMapCallBackProcPtr BitMapCallBackUPP;
#endif

pascal
CMError
CWMatchBitMap(CWorld myCWorld, const CMBitMap *bitMap,
              BitMapCallBackUPP progressProc,
              void *refCon, CMBitMap *matchedBitMap);
```

CWCheckBitMap

Gamut test pixel data of `bitMap` according to the `Profiles` corresponding to `myCWorld`.

```
pascal
CMError
CWCheckBitMap(CWorld myCWorld, CMBitMap *bitMap,
              BitMapCallBackUPP progressProc,
              void *refCon,
              CMBitMap *resultBitMap);
```

Quickdraw-specific Matching

CWMatchPixMap

Match pixel data of `myPixMap` according to the `Profiles` corresponding to `myCWorld`. `myPixMap` must be non-relocatable.

```
pascal
CMError
CWMatchPixMap(CWorld myCWorld, PixMap *myPixMap,
              BitMapCallBackUPP progressProc, void *ref-
Con);
```

CWCheckPixMap

Gamut-test pixel data of `myPixMap` according to the `Profiles` correspond-

ing to myCWorld. myPixMap must be non-relocatable.

```
pascal
CMError
CWCheckPixMap(CWorld myCWorld, PixMap *myPixMap,
              BitMapCallBackUPP progressProc, void *refCon,
               BitMap *resultBitMap);
```

NBeginMatching

Set up a high-level matching session forsrc and dst Profiles.

If the currentGDevice is a monitor device then subsequent Quickdraw operations to all monitorGDevices will matched. Otherwise automatic matching will be limited to the currentGDevice.

```
pascal
CMError
NBeginMatching(Profile src, Profile dst, CMMatchRef
*myRef)
```

EndMatching

Restore unmatched Quickdraw operations and free private allocations for the
high-level matching session.

```
pascal
void
EndMatching(CMMatchRef myRef)
```

NDrawMatchedPicture

Bracket DrawPicture(_) with WhiteFang function calls which set up and take down a high-level matching session. The Picture will be drawn with matched colors to all monitorGDevices. If the current GDevice is not a monitor Device then matching will occur for that GDevice only. This function and DrawPicture(_) operate in the context of the currentGrafPort. Color-matching PicComments embedded in the Picture are respected.

```
pascal
   void
   NDrawMatchedPicture(PicHandle myPicture, Profile dst,
                       Rect *myRect)
```

EnableMatching

Generate a CMEnableMatching or CMDisableMatching PicComment for currently open Picture.

```
pascal
```

```
void
EnableMatching(Boolean enableIt)
```

External Profile Searching

Search and access functions for Profiles in the ColorSync™ Profiles folder. On the Macintosh this is located in {SystemFolder}Preferences: Search specifications are defined by :

```
/* Defines for version 2.0 ProfileSearchRecord.searchMask */
#define kMatchProfileCMMType0x00000001
#define kMatchProfileClass0x00000002
#define kMatchDeviceSpace0x00000004
#define kMatchInterchangeSpace0x00000008
#define kMatchManufacturer0x00000010
#define kMatchModel  0x00000020
#define kMatchAttributes0x00000040
#define kMatchProfileFlags0x00000080

typedef
pascal
Boolean (*ProfileFilterProcPtr)(Profile prof, void *refCon);

#if USESROUTINEDESCRIPTORS
typedef UniversalProcPtr ProfileFilterUPP;
#else
typedef ProfileFilterProcPtr ProfileFilterUPP;
#endif

struct ProfileSearchRecord {
   OSType       CMMType;
   OSType       profileClass;
   OSType       deviceColorSpace;
   OSType       interchangeColorSpace;
   unsigned longdeviceManufacturer;
   unsigned longdeviceModel;
   unsigned longdeviceAttributes[2];
   unsigned longprofileFlags;

   unsigned longsearchMask;
   ProfileFilterUPPfilter;
};
typedef struct ProfileSearchRecordProfileSearchRecord;
```

The bits of searchMask specify corresponding fields which must match in the search.

filter is a client function which can be used to search on elements outside the defined ProfileSearchRecord fields to implement OR or AND search logic. This function returns true if the specified profile is to be filtered from the search.

The data structure which contains the result of a profile search is a private abstract type.

```
typedef structPrivateProfileSearchResult*ProfileSearch;
```

The search functions are implemented in a private standalone Component. It is registered at startup with the following parameters.

componentType	'prof'
componentSubType	'dflt'
componentManufacturer	'appl'

This Component is opened and called by the resident dispatcher to implement the WhiteFang API.

NewProfileSearch

Description.

```
pascal
CMError
NewProfileSearch(ProfileSearchRecord *searchSpec,
                 void *refCon, unsigned long *count,
                 ProfileSearch *searchResult);
```

UpdateProfileSearch

Update an existing search result. This is necessary if the contents of the ColorSync™ Profiles folder has changed since the search result was created.

Sharing a disk over a network makes it possible for the folder contents to be modified at any time.

```
pascal
CMError
UpdateProfileSearch(ProfileSearch search, void *refCon,
                    unsigned long *count)
```

DisposeProfileSearch

Free private allocations associated with a `search result`.

```
pascal
void
DisposeProfileSearch(ProfileSearch search);
```

SearchGetIndexedProfile

Open and return reference to the `Profile` at `index` into search result. The caller is responsible for closing the returned `Profile`.

```
pascal
CMError
SearchGetIndexedProfile(ProfileSearch search, unsigned
long index,
                        Profile *prof);
```

SearchGetIndexedProfileFileSpec

Get file specification for the `Profile` at `index` into search result.

Cross-platform implementations will define the `ProfileFile` parameter as appropriate to the OS.

```
pascal
CMError
```

```
SearchGetIndexedProfileFileSpec(ProfileSearch search,
                                unsigned long index,
                                FSSpec *profileFile);
```

System Profile Access

The System Profile represents an abstract device. It is used as the default color space if a Profile is not specified for the WhiteFang matching functions. The System Profile is that of an RGB or GRAY monitor for Color QuickDraw.

GetSystemProfile

Return Profile reference for current System Profile. Caller is responsible for calling `CloseProfile(...)` when finished.

```
pascal
CMError
GetSystemProfile(Profile *prof);
```

SetSystemProfile

Set specified profile file as the current System Profile.

```
pascal
CMError
SetProfile(const FSSpec *profileFileSpec);
```

Utilities

GetColorSyncFolderSpec

Return the HFS volume refNum and directory ID of the ColorSync™ Profiles folder. Optionally create folder if it does not already exist.

```
pascal
CMError
GetColorSyncFolderSpec(short vRefNum, Boolean createFold-
er,
                       short *foundVRefNum, long *foundDi-
rID);
```

GetCWInfo

Supply information about an existing CWorld. The information is returned in the caller's CWInfoRecord structure whose address is passed as the `info` parameter.

```
struct CMMInfoRecord {
   OSType       CMMType;
   long                 CMMVersion;
};
```

```
typedef struct CMMInfoRecordCMMInfoRecord;

struct CWInfoRecord {
   unsigned shortcmmCount;
   CMMInfoRecordcmmInfo[2];
};

typedef struct CWInfoRecordCWInfoRecord;


pascal
CMError
GetCWInfo(CWorld myCWorld, CWInfoRecord *info);
```

PostScript support functions

Three functions are added to the WhiteFang API and as optional functions in the CMM Component API to support color-matching by PostScript Level 2 devices.

The flags parameter to these three function specifies the format of the PostScript data to be returned.

```
enum {
   ps7bit = 1,
   ps8bit = 2
};
```

`ps7bit` specifies 7-bit ASCII

`ps8bit` specifies 8-bit per character data as specified in the *PostScript Language Reference Manual* second edition.

GetPS2ColorSpace

Return ASCII text usable as the parameter to the PostScript `setcolorspace` operator.
This operator characterizes the color space of subsequent graphics data.

This function is dispatched to the CMM Component corresponding to `srcProf`. If the Component is not available or the function is not implemented then it is dispatched to the default CMM `textData` is set by the default CMM and the function result is `cmMethodNotFound`.

A caller-supplied FlattenProc function is used to perform the actual data transfer.

```
pascal
CMError
GetPS2ColorSpace(Profile srcProf, unsigned long flags,
                 FlattenUPP proc, void *refCon);
```

Parameters
srcProf (in) Reference to source Profile.

flags (in) specifies desired format of PostScript data

proc (in) Function for data transfer

refCon (in) Client data which is passed as a parameter to calls to proc.

Result codes
noErr
CMMethodNotFound CMM corresponding to srcProf not available.

System or ColorSync result code if an error occurs.

GetPS2ColorRenderingIntent

Return ASCII text usable as the parameter to the PostScript setRendering-Intent operator. This operator specifies the color-matching option for subsequent graphics data.

This function is dispatched to the CMM Component corresponding to srcProf. If the Component is not available or the function is not implemented then it is dispatched to the default CMM textData is set by the default CMM and the function result is cMMethodNotFound.

A caller-supplied FlattenProc function is used to perform the actual data transfer.

```
pascal
CMError
GetPS2ColorRenderingIntent(Profile srcProf, unsigned long
flags,
                           FlattenUPP proc, void *refCon);
```

Parameters
srcProf (in) Reference to source Profile.

flags (in) specifies desired format of PostScript data

proc (in) Function for data transfer

refCon (in) Client data which is passed as a parameter to calls to proc.

Result codes
noErr
CMMethodNotFound CMM corresponding to prof not available.

System or ColorSync result code if an error occurs.

GetPS2ColorRendering

Return ASCII text usable as the parameter to the PostScript `setColorRendering` operator. This operator specifies the PostScript Color Rendering dictionary to be used for the following graphics data.

This function is dispatched to the CMM Component corresponding to `srcProf`. If the Component is not available or the function is not implemented then it is dispatched to the default CMM `textData` is set by the default CMM and the function result is `cmMethodNotFound`.

A caller-supplied FlattenProc function is used to perform the actual data transfer.

```
pascal
CMError
GetPS2ColorRendering(Profile srcProf, Profile dstProf,
                     unsigned long flags, FlattenProc
proc,
                     void *refCon);
```

Parameters

srcProf (in) Reference to source `Profile`.

dstProf (in) Reference to destination `Profile`.

flags (in) specifies desired format of PostScript data

proc (in) Function for data transfer

refCon (in) Client data which is passed as a parameter to calls to `proc`.

Result codes

`noErr`

`cmMethodNotFound` CMM corresponding to `prof` not available.

System or ColorSync result code if an error occurs.

CMM Component Function Reference

A CMM is a Component Manager Component. The standard required Component functions must be supported.

kComponentOpenSelect
kComponentCloseSelect
kComponentCanDoSelect
kComponentVersionSelect

The standard optional Component functions may also be supported

kComponentRegisterSelect
kComponentTargetSelect

The dispatcher determines the CMM Components which will be responsible from the `CMMType` element of the `Profile` parameters to the client call. One or two CMMs perform the actual work.

A separate ComponentInstance is opened for each matching session.The set of `Profiles` is unique for each session. The Component should allocate private storage to store the necessary information for the instance and use Component Manager functions to manage the storage. A Component should free the storage in the ComponentClose function.

The CMM Component functions are in either required or optional categories. For optional functions, the dispatcher will call the Component with the kComponentCanDoSelect selector before calling the actual function.

If the CMM does not support an optional function, the dispatcher will use an alternative to complete the client request. For example, if the CMM does not support the CMMatchBitMap function, the dispatcher will convert the BitMap data `CMColor` format and call the CMM`CMMatchColors` function. The matched colors will then be copied to the client BitMap.

Required Functions

NCMInit

Component function called to initialize a matching session. Called after the Component has been opened with OpenComponent(...).

```
pascal
CMError
NCMInit(ComponentInstance CMSession, Profile srcProfile,
        Profile dstProfile);
```

CMMatchColors

Color-matching on a list of CMColor. The source and destination data types are specified by the `Profile` parameters to the previous call to the NCMInit(...), CMInit(...), or CMConcatInit(...) function.

```
pascal
CMError
CMMatchColors(ComponentInstance CMSession,
              CMColorList myColors, long count);
```

CMCheckColors

Gamut test a list of CMColor. The source and destination are specified by the `Profile` parameters to the previous call to the NCMInit(...), CMInit(...), or CMConcatInit(...) function.

```
pascal
CMError
CMCheckColors(ComponentInstance CMSession,
              CMColorList myColors, long count,
              CMGamutResult result);
```

CMValidateProfile

Test whether `Profile` contains the minimum set of elements required by this CMM.

Third-party CMMs should first call the default CMM using Component Manager functions to assure that the minimum default elements are present.

```
pascal
CMError
CMValidateProfile(ComponentInstance CMSession, Profile
prof,
                  Boolean *valid);
```

CMFlattenProfile

Flatten specified `Profile` to the external `CMProfile` format. A caller-supplied `FlattenProcUPP` is called to perform the actual data transfer. The `flags` parameter specifies options for the content of the flattened `CMProfile`.

The default Apple implementation of this function must be capable of flattening any `Profile` which contains the minimum required elements.

```
pascal
CMError
CMFlattenProfile(ComponentInstance CMSession, Profile
prof,
                 long flags, FlattenUPP proc, void *ref-
Con);
```

CMUnflattenProfile

Unflatten an external format `CMProfile` to an internal format `Profile` reference.

This function may be used to create a `Profile` reference from a `CMProfile` embedded in a graphic document. The caller is responsible for calling `CloseProfile(...)` when the use of the `Profile` reference is completed.

The returned `Profile` has a temporary backing file if necessary. The file is deleted at the time `CloseProfile(...)` is called.

```
pascal
CMError
CMUnflattenProfile(ComponentInstance CMSession, Profile
*prof,
                    FlattenUPP proc, void *refCon)
```

Parameters
CMSession(in)Reference to `ComponentInstance`.

prof (out) Returns `Profile` reference if function result is `noErr`. Otherwise undefined.

proc (in) Function for data transfer

refCon (in) Client data which is passed as a parameter to calls to `proc`.

Result codes
noErr

System or ColorSync result code if an error occurs

Optional Functions

CMMatchBitMap

Match pixel data of bitMap according to the `Profile` parameters supplied to a previous call to `CMInit(...)`, `NCMInit(...)`, or `CMConcatInit(...)`.

```
pascal
CMError
CMMatchBitMap(ComponentInstance CMSession, const CMBitMap
*bitMap,      BitMapCallBackUPP progressProc, void *refCon,
               CMBitMap *matchedBitMap);
```

CMCheckBitMap

Gamut test pixel data of bitMap according to the `Profile` parameters supplied to a previous call to `CMInit(...)`, `NCMInit(...)`, or `CMConcatInit(...)`.

```
pascal
CMError
CMCheckBitMap(ComponentInstance CMSession, const CMBitMap
*bitMap,
              BitMapCallBackUPP progressProc, void *refCon,
               CMBitMap *resultBitMap);
```

CMMatchPixMap

Match pixel data of `myPixMap` according to the `Profile` parameters supplied to a previous call to `CMInit(…)`, `NCMInit(…)`, or `CMConcatInit(…)`. `myPixMap` is non-relocatable.

```
pascal
CMError
CMMatchPixMap(ComponentInstance CMSession, PixMap *myPix-
Map,
              BitMapCallBackUPP progressProc, long refCon);
```

CMCheckPixMap

Gamut-test pixel data of `myPixMap` according to the `Profile` parameters supplied to a previous call to `CMInit(…)`, `NCMInit(…)`, or `CMConcatInit(…)`. `myPixMap` is non-relocatable..

```
pascal
CMError
CMCheckPixMap(ComponentInstance CMSession, PixMap *myPix-
Map,
               BitMapCallBackUPP progressProc, BitMap *my-
BitMap,
               long refCon);
```

CMConcatInit

This is a special purpose function that is outside the domain of the standard source → destination matching session. A single CMM is specified by the caller which is responsible for the entire session. That CMM may use Component Manager functions to call other CMMs if required.

The processing possible with this function can be complex, including operations outside of simple "matching".

Note that the `dataType` and `deviceType` of the first and last `Profiles` must be valid device values. Intermediate `Profiles` may be non-device type values.

```
pascal
CMError
CMConcatInit(ComponentInstance CMSession,
             ConcatProfileSet *profileSet);
```

Parameters

CMSession(in)Reference to ComponentInstance.

profileSet(in) `ConcatProfileSet` contains an array of `Profiles` which describe the processing to be carried out.

The `profileSet array is` in processing order— Source through Destination. A minimum of one `Profile` must be specified.

The `CMMType` element of `Profile` at `keyIndex` specifies this CMM, which is entirely responsible for the session.

```
struct ConcatProfileSet
{
  unsigned shortkeyIndex;
  unsigned shortcount;
  Profile   profileSet[1];
};
```

Result codes

`noErr`

System or ColorSync result code if an error occurs.

CMNewLinkProfile

This function allows the creation of a new `Profile` which embodies the transforms included by `NCWConcatColorWorld(_)`.

Note that the `dataType` and `deviceType` of the first and last `Profiles` must be valid device values. Intermediate `Profiles` may be non-device type values.

```
pascal
CMError
CMNewLinkProfile(ComponentInstance CMSession, Profile
*prof,
                 const FSSpec *backingFileSpec,
                 ConcatProfileSet *profileSet);
```

Parameters

CMSession(in)Reference to ComponentInstance.

backingFileSpec (in)File spec for the resulting `CMProfile`.

profileSet(in) `ConcatProfileSet` contains an array of `Profiles` which describe the processing to be carried out.

The `profileSet array is` in processing order— Source through Destination. A mini-

mum of one `Profile` is required.

The `CMMType` element of `Profile` at `keyIndex` specifies the CMM which is entirely responsible for the session.

Result codes

`noErr`

System or ColorSync result code if an error occurs.

CMGetPS2ColorSpace

Return ASCII text usable as the parameter to the PostScript `setcolorspace` operator.
This operator characterizes the color space of subsequent graphics data.

A caller-supplied FlattenProc function is used to perform the actual data transfer.

```
pascal
CMError
CMGetPS2ColorSpace(ComponentInstance CMSession, Profile
srcProf,           unsigned long flags, FlattenUPP proc,
                   void *refCon);
```

Parameters
CMSession(in)Reference to ComponentInstance.

srcProf(in) Reference to source `Profile`.

flags (in) specifies desired format of PostScript data

proc (in) Function for data transfer

refCon (in) Client data which is passed as a parameter to calls to `proc`.

Result codes
`noErr`

System or ColorSync result code if an error occurs.

CMGetPS2ColorRenderingIntent

Return ASCII text usable as the parameter to the PostScript `setRendering-Intent` operator. This operator specifies the color-matching option for subsequent graphics data.

A caller-supplied FlattenProc function is used to perform the actual data transfer.

```
pascal
CMError
CMGetPS2ColorRenderingIntent(ComponentInstance CMSession,
                             Profile srcProf,
                             unsigned long flags,
                             FlattenUPP proc, void *ref-
Con);
```

Parameters

CMSession(in)Reference to ComponentInstance.

srcProf(in) Reference to source Profile.

flags (in) specifies desired format of PostScript data

proc (in) Function for data transfer

refCon (in) Client data which is passed as a parameter to calls to proc.

Result codes
noErr

System or ColorSync result code if an error occurs.

CMGetPS2ColorRendering

Return ASCII text usable as the parameter to the PostScript setColorRendering operator. This operator specifies the PostScript Color Rendering dictionary to be used for subsequent graphics data.

A caller-supplied FlattenProc function is used to perform the actual data transfer.

```
pascal
CMError
CMGetPS2ColorRendering(ComponentInstance CMSession, Pro-
file srcProf,
                        Profile dstProf, unsigned long
flags,
                        FlattenUPP proc, void *refCon);
```

Parameters
CMSession(in)Reference to ComponentInstance.

srcProf(in) Reference to source Profile.

dstProf(in) Reference to destination Profile.

flags (in) specifies desired format of PostScript data

proc (in) Function for data transfer

refCon (in) Client data which is passed as a parameter to calls to proc.

Result codes
noErr

System or ColorSync result code if an error occurs

What Is Claimed Is:

1. In a color management system providing a modular framework having a default color matching module and in which color matching modules may be added to and removed from the system, a method of selecting and using one or more color matching modules to perform a requested color matching operation of source data specified in a source data space described by a source profile and destination data specified in a destination data space described by a destination profile, the source profile identifying a designated source-profile-preferred color matching module and the destination profile identifying a designated destination-profile-preferred color matching module, the method comprising the steps of:

a) if the source profile preferred color matching module is unavailable, designating instead a default color matching module as the designated source-profile-preferred color matching module for purposes of selection, and if the destination profile preferred color matching module is unavailable, designating instead a default color matching module as the designated destination-profile-preferred color matching module for purposes of selection,

b) if following step a) the designated source-profile-preferred color matching module and the designated destination-profile-preferred color matching module are the same color matching module, using said same color matching module to build a concatenated transformation table by sampling color points in a source color space specified in the source profile and converting the sampled color points into a destination color space specified in the destination profile, and converting colors from the source color space to the destination color space by table lookup using the concatenated transformation table; else

c) if following step a) the designated source-profile-preferred color

matching module and the designated destination-profile-preferred color matching module are different, using the designated source-profile-preferred color matching module and the designated destination-profile-preferred color matching module to build a concatenated transformation table by sampling color points in a source color space specified in the source profile and converting the sampled color points into a destination color space specified in the destination profile, and converting colors from the source color space to the destination color space by table lookup using the concatenated transformation table.

2. The method of Claim 1, wherein the designated source-profile-preferred color matching module and the designated destination-profile-preferred color matching module use a common color interchange space, step c) further comprising:

converting the source data to data in the common color interchange space using the designated source-profile-preferred color matching module; and

converting the data in the common color interchange space to destination data using the designated destination-profile-preferred color matching module.

3. The method of Claim 1, wherein the designated source-profile-preferred color matching module and the designated destination-profile-preferred color matching module use different color interchange spaces, step c) further comprising:

converting the source data to data in the source color interchange space using the designated source-profile-preferred color matching module;

converting the data in the source color interchange space to a destination color interchange space using a color space conversion function; and

converting the data in the data color interchange space to destination data using the designated destination-profile-preferred color matching module.

4. In a color management system providing a modular framework having a default color matching module and in which color matching modules may be added to and removed from the system, a method of validating color profiles describing color characteristics of color images or devices, comprising the steps of:

accessing the profile to identify a preferred color matching module;

if the preferred color matching module is available, dispatching the preferred color matching module to perform validation of the profile; and

if the preferred color matching module is not available, dispatching a default color matching module to perform validation of the profile.

5. In a color management system providing a modular framework having a default color matching module and in which color matching modules may be added to and removed from the system, a method of converting data in a color profile designating a preferred color matching module to color profile PostScript data, comprising the steps of:

if the preferred color matching module is available, dispatching the preferred color matching module to convert the non-PostScript data to PostScript data; and

if the preferred color matching module is not available, dispatching a default color matching module to convert the non-PostScript data to PostScript data.

6. In a color management system providing a modular framework having a default color matching module and in which color matching modules may be added to and removed from the system, a method of creating a

concatenation of color profiles describing color characteristics of color devices, comprising the steps of:

designating one of the color profiles as a key profile, the key profile containing data identifying a preferred color matching module;

if the preferred color matching module is available, dispatching the preferred color matching module to perform the concatenation of the color profiles; and

if the preferred color matching module is not available, dispatching a default color matching module to perform the concatenation of the color profiles.

7. In a color management system providing a modular framework having a default color matching module and in which color matching modules may be added to and removed from the system, a method of creating a concatenation of a sequence of color profiles, including a first source profile and a last destination profile, describing color characteristics of color devices, each of the color profiles identifying a designated profile-preferred color matching module, the method comprising the steps of:

if a designated profile-preferred color matching module is unavailable, designating instead a default color matching module as the designated profile-preferred color matching module; and

building a concatenated transformation table by sampling color points in a source color space specified in a first profile and converting the sampled color points into a final color space specified in a final profile using the designated profile-preferred color matching modules by:

for a first color profile, converting color profile information from a source profile color space to a color interchange space using a corresponding designated profile-preferred color matching module;

for each successive color profile before a last color profile, converting from a color interchange space of a preceding color matching module to a color interchange space of a present color matching module

if the color interchange space of the preceding color matching module and the color interchange space of the present color matching module are different, converting from the color interchange space of the present color matching module to a color profile space of the present color matching module using a corresponding designated profile-preferred color matching module, and converting from the color profile space of the present color matching module to a color interchange space of the present color matching module using said corresponding designated profile-preferred color matching module; and

for a last color profile, converting from a color interchange space of a preceding color matching module to a color interchange space of the last color matching module if the color interchange space of the preceding color matching module and the color interchange space of the last color matching module are different; and

converting from the color interchange space of the last color matching module to a color profile space of the last color matching module using a corresponding designated profile-preferred color matching module and converting colors from the first color space to the last color space by table lookup using the concatenated transformation table.

8. In a color management system providing a modular framework having a default color matching module and in which color matching modules may be added to and removed from the system, a method of creating a device link profile describing an overall color characteristic of multiple color devices, comprising the steps of:

if a designated profile-preferred color matching module is unavailable, designating instead a default color matching module as the designated profile-preferred color matching module; and

building a concatenated transformation table by sampling color points in

a source color space specified in a first profile and converting the sampled color points into a final color space specified in a final profile using the designated profile-preferred color matching modules by:

for a first color profile, converting color profile information from a source profile color space to a color interchange space using a corresponding designated profile-preferred color matching module;

for each successive color profile before a last color profile, converting from a color interchange space of a preceding color matching module to a color interchange space of a present color matching module if the color interchange space of the preceding color matching module and the color interchange space of the present color matching module are different, converting from the color interchange space of the present color matching module to a color profile space of the present color matching module using a corresponding designated profile-preferred color matching module, and converting from the color profile space of the present color matching module to a color interchange space of the present color matching module using said corresponding designated profile-preferred color matching module; and

for a last color profile, converting from a color interchange space of a preceding color matching module to a color interchange space of the last color matching module if the color interchange space of the preceding color matching module and the color interchange space of the last color matching module are different, and converting from the color interchange space of the last color matching module to a color profile space of the last color matching module using a corresponding designated profile-preferred color matching module; and

creating a new color profile including the concatenated transformation table.

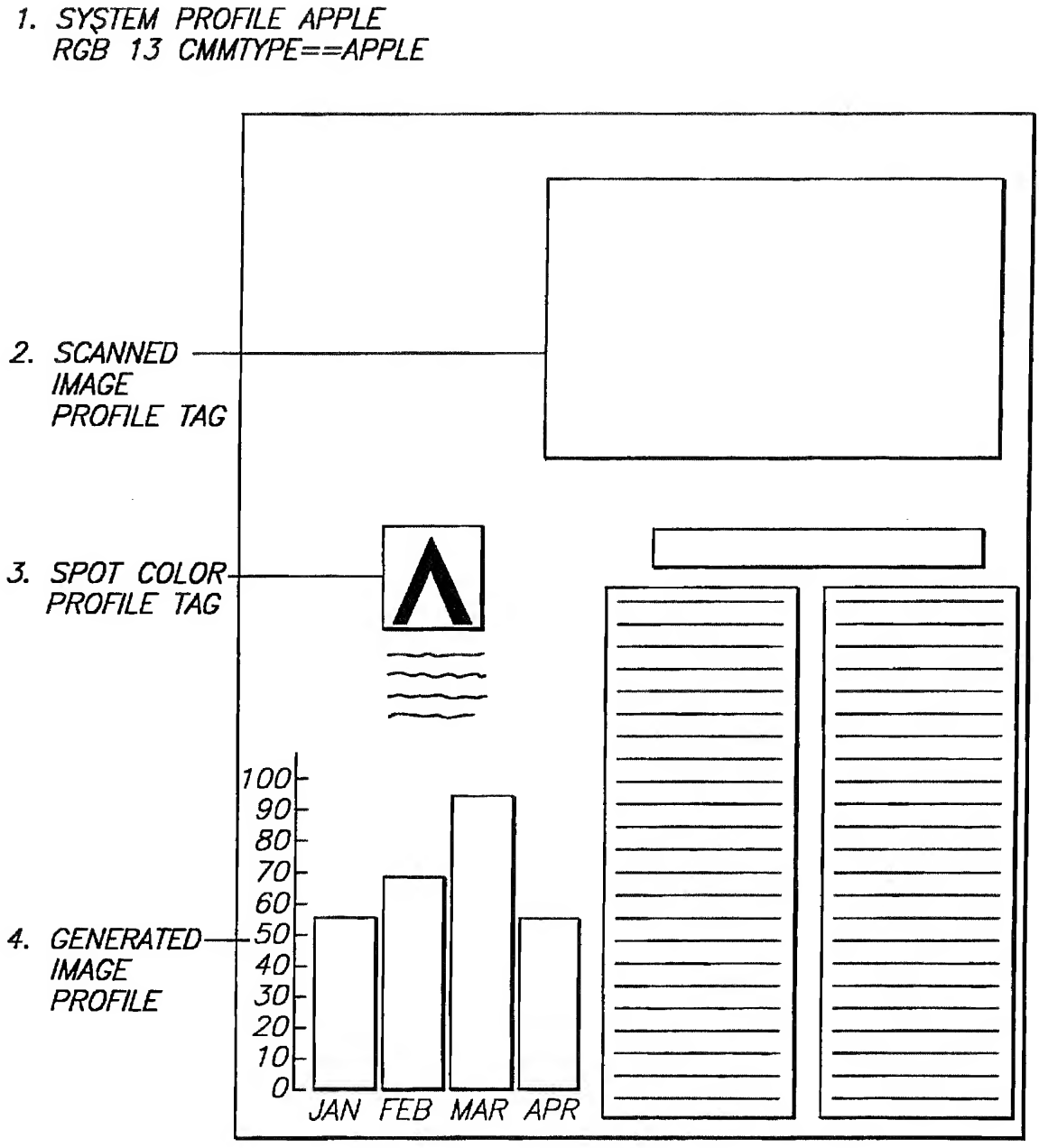

1. SYSTEM PROFILE APPLE
RGB 13 CMMTYPE==APPLE
2. SCANNED IMAGE PROFILE TAG
3. SPOT COLOR PROFILE TAG
4. GENERATED IMAGE PROFILE
100
90
80
70
60
50
40
30
20
10
0
JAN
FEB
MAR
APR

FIG. 1

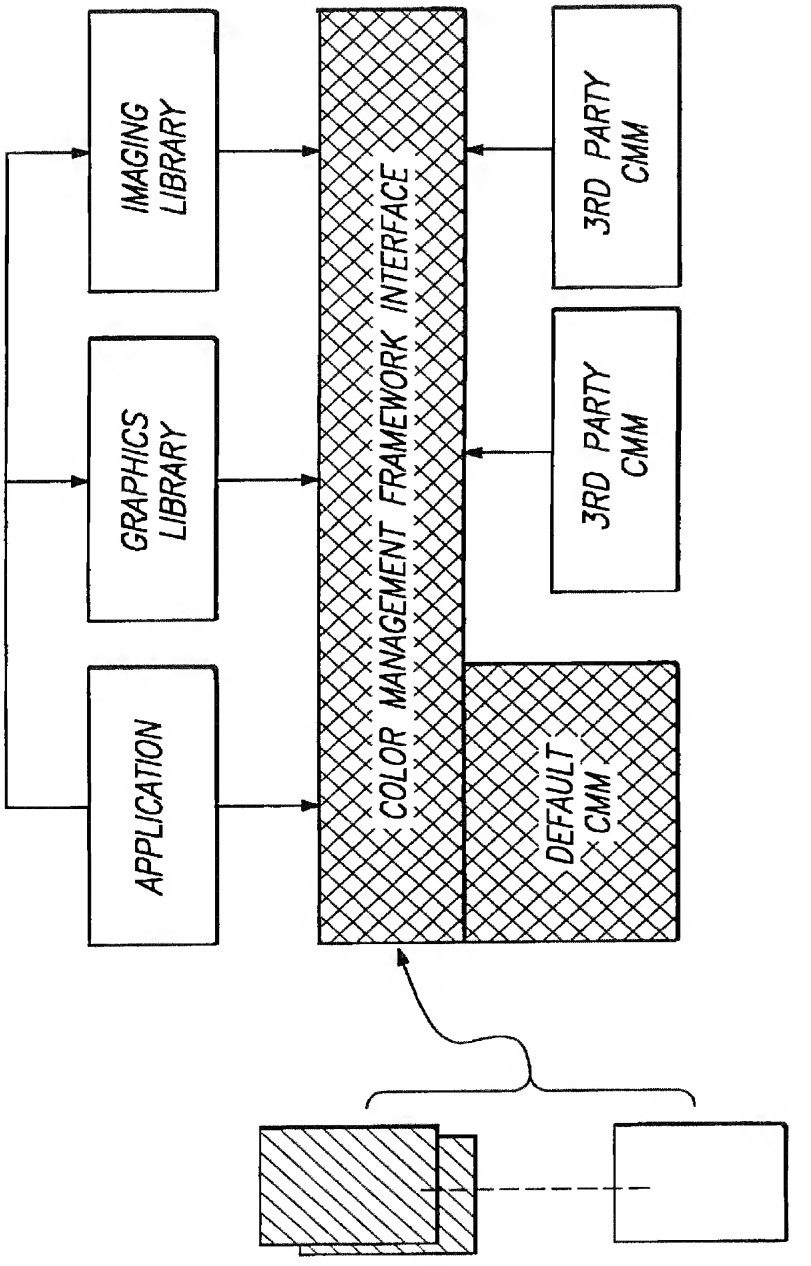


FIG. 2
PRIOR ART

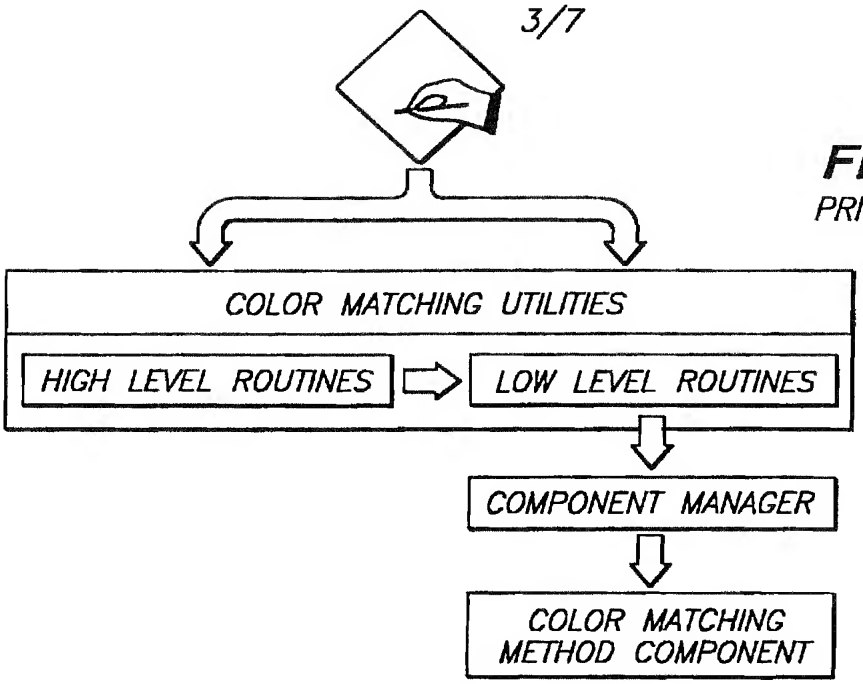


FIG. 3
PRIOR ART

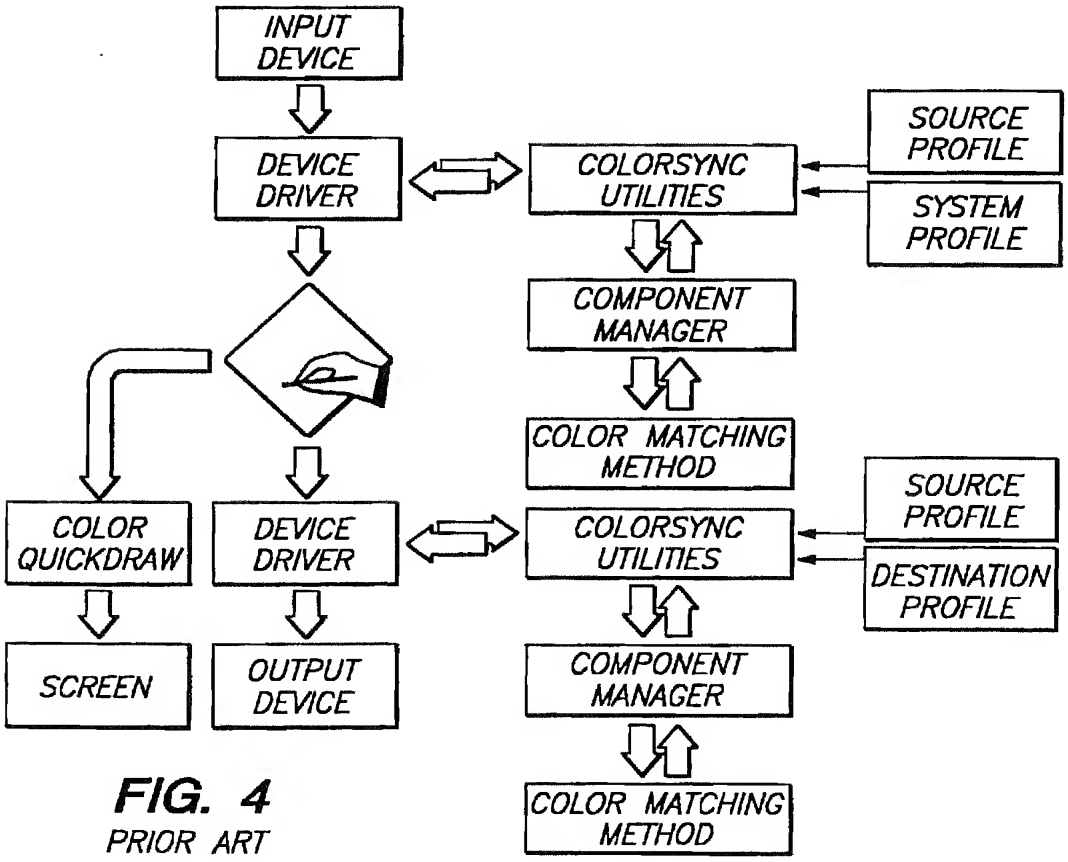


FIG. 4
PRIOR ART

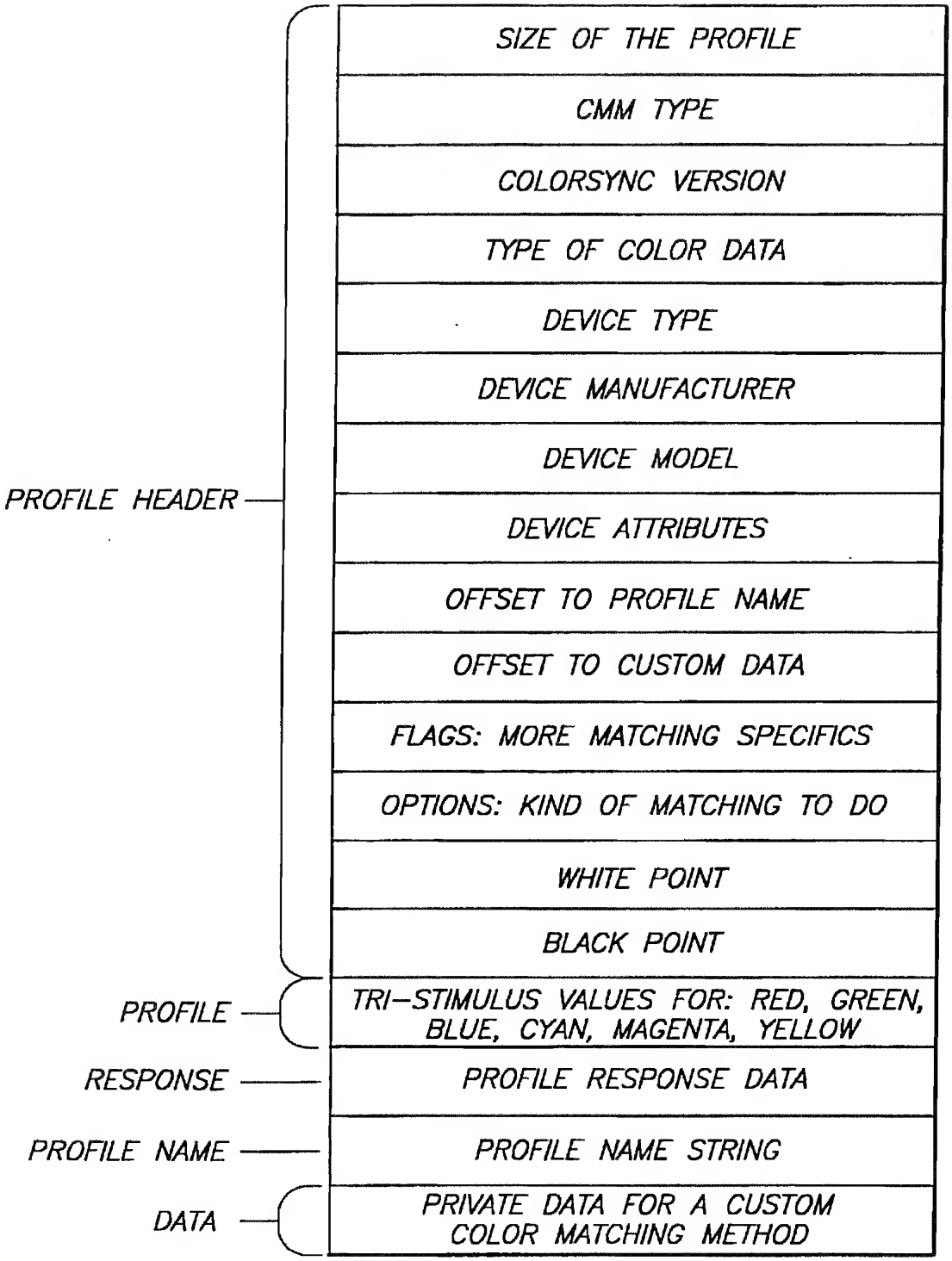


FIG. 5
PRIOR ART

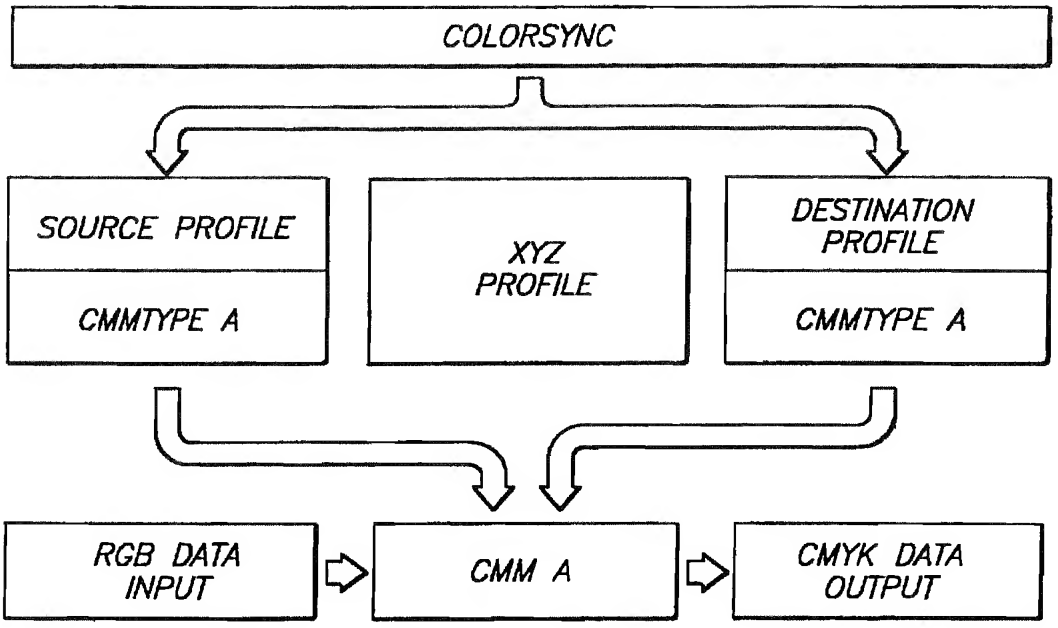


FIG. 6 PRIOR ART

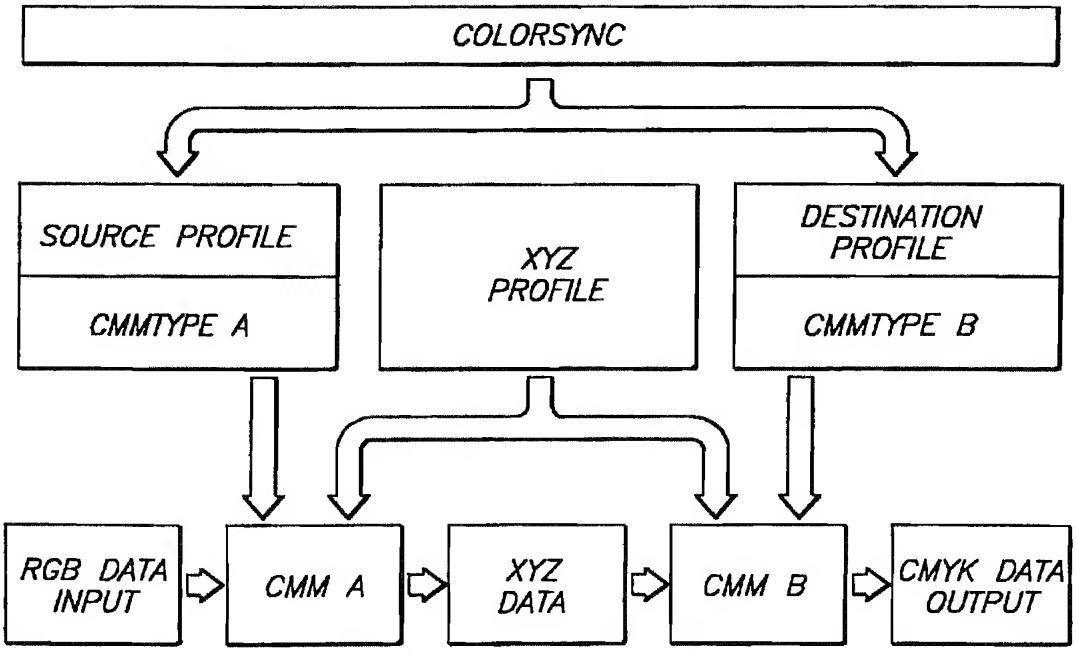


FIG. 7 PRIOR ART

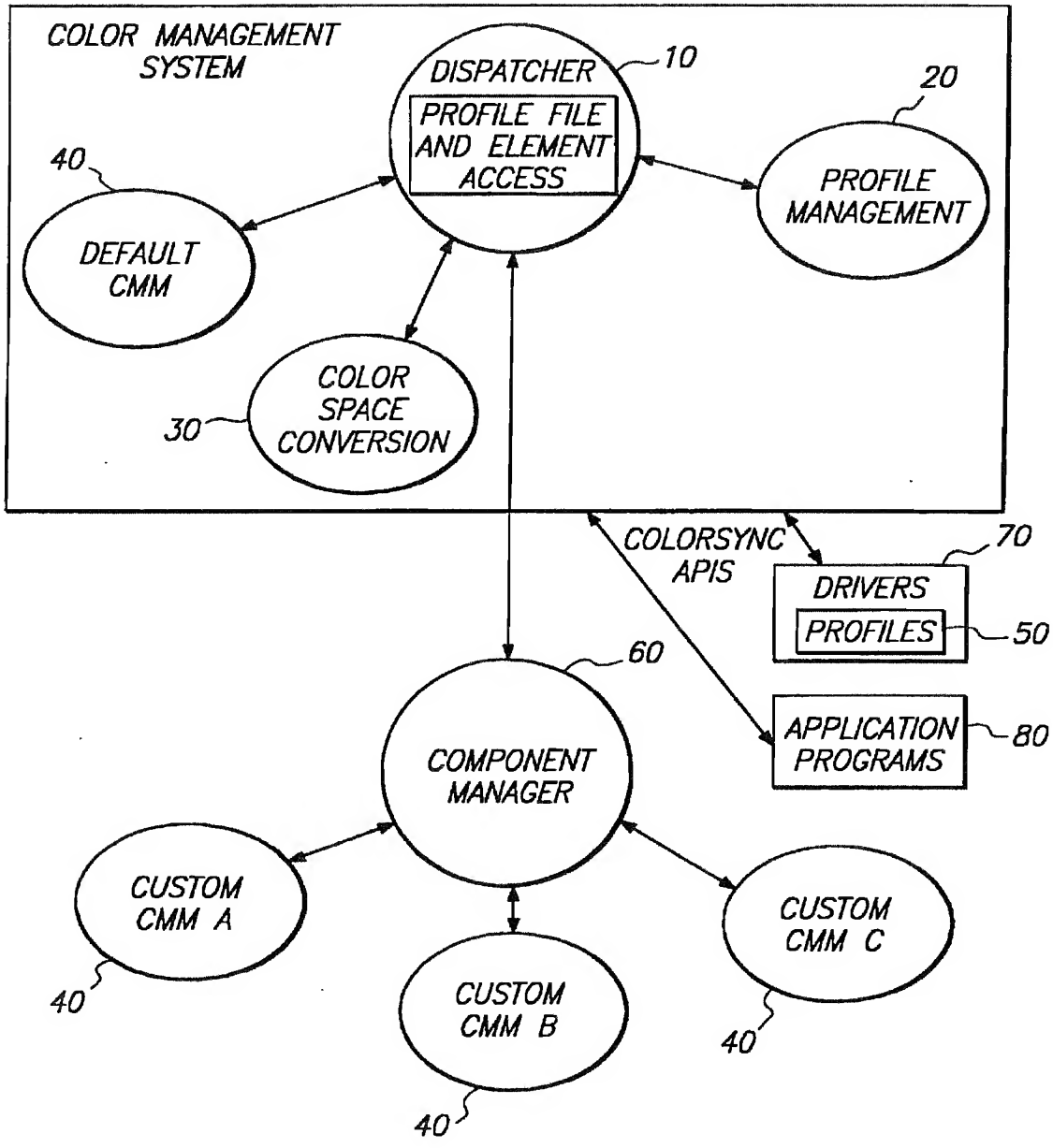

COLOR MANAGEMENT SYSTEM
DISPATCHER
10
PROFILE FILE AND ELEMENT ACCESS
20
PROFILE MANAGEMENT
40
DEFAULT CMM
COLOR SPACE CONVERSION
30
COLORSYNC APIS
70
DRIVERS
PROFILES
50
60
COMPONENT MANAGER
APPLICATION PROGRAMS
80
CUSTOM CMM A
40
CUSTOM CMM B
40
CUSTOM CMM C
40

FIG. 8

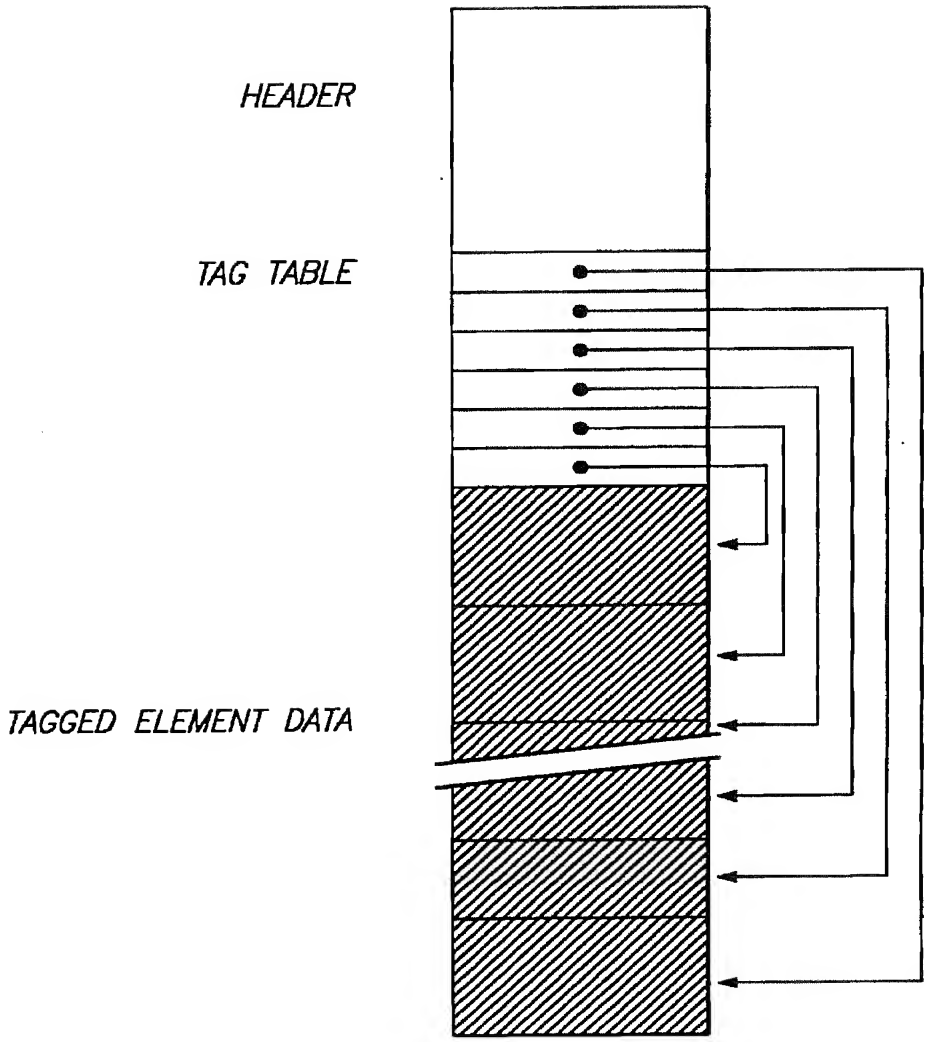


FIG. 9

INTERNATIONAL SEARCH REPORT

Interna. al Application No
PCT/US 95/06237

A. CLASSIFICATION OF SUBJECT MATTER
IPC 6 G06T11/00

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 6 G06T

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US,A,5 233 684 (ULICHNEY ROBERT A) 3 August 1993 see claim 1	1-8
A	EP,A,0 594 370 (XEROX CORP) 27 April 1994	
A	ACM TRANSACTIONS ON GRAPHICS, vol. 11, no. 4, 1 October 1992 pages 373-405, XP 000335146 KASSON J M ET AL 'AN ANALYSIS OF SELECTED COMPUTER INTERCHANGE COLOR SPACES'	
A	EP,A,0 475 554 (SCITEX CORP LTD) 18 March 1992	

☐ Further documents are listed in the continuation of box C.

☒ Patent family members are listed in annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

1

Date of the actual completion of the international search	Date of mailing of the international search report
21 September 1995	05. 10. 95

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+ 31-70) 340-2040, Tx. 31 651 epo nl, Fax (+ 31-70) 340-3016	Perez Molina, E

INTERNATIONAL SEARCH REPORT

Information on patent family members

Internal Application No
PCT/US 95/06237

Patent document cited in search report	Publication date	Patent family member(s)		Publication date
US-A-5233684	03-08-93	NONE		
EP-A-0594370	27-04-94	US-A-	5384901	24-01-95
		JP-A-	6139365	20-05-94
EP-A-0475554	18-03-92	CA-A-	2035658	06-08-91
		EP-A-	0449407	02-10-91
		JP-A-	5153380	18-06-93
		CA-A-	2059193	16-07-92
		EP-A-	0495563	22-07-92
		JP-A-	4316848	09-11-92